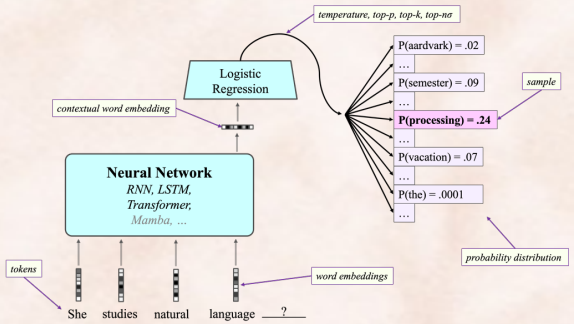
ITCS 6101/8101: Natural Language Processing

Language Modeling Architectures



Razvan C. Bunescu

Department of Computer Science @ CCI

rbunescu@charlotte.edu

Outline

- **Language modeling definition**:
  - Causal vs. Masked language modeling.
- **N-gram language models**.
- **Neural language models**.
  - Encoder, Decoder, Encoder-Decoder architectures.
  - FCNs → RNNs → Transformer
- **Large Language Models (LLM) basics**:
  - Autoregressive Generation and Training Scheme.

- **The AI Inverse Problem**:
  - From predicting words to AI.
- **What is Intelligence?**

Language Modeling (LM)

- **Causal** Language Modeling:
  - **Predict the next word in a sequence**:
    - AI systems use machine _____
      - eat?
      - learning?
      - frogs?
      - …
    - **The LM estimates P(word | $word_{-1}$, $word_{-2}$, ...)**
      - we want P(learning | machine, use) >> P(about | machine, eat).
  - **Decoder** neural architectures are widely used to train LMs:
    - GPT, Gemini, Llama, Grok, Mixtral, Claude, …

Language Modeling (LM)

- **Masked** Language Modeling:
  - **Predict the most likely word in a context**:
    - AI systems use machine _____ models for language understanding .

      eat?

      learning?

      frogs?

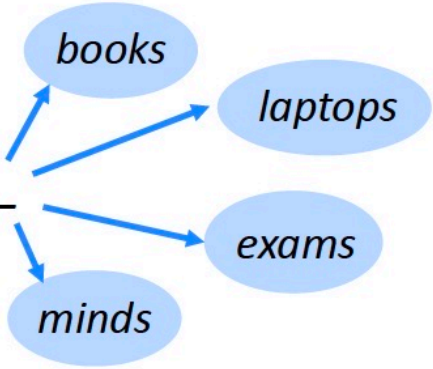
      …
    - The LM estimates P(word | $word_{-1}$, $word_{-2}$, ...; $word_1$, $word_2$, ...)
      - we want P(learning | machine, use; models, for) >> P(frogs | machine, use; models, for).
  - **Encoder** neural architectures are used to train masked LMs.
    - BERT, RoBERTa, DistilBERT, …

Language Modeling

white slides selected from cs224n @ Stanford

- **Language Modeling** is the task of predicting what word comes next.

the students opened their _____



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text.

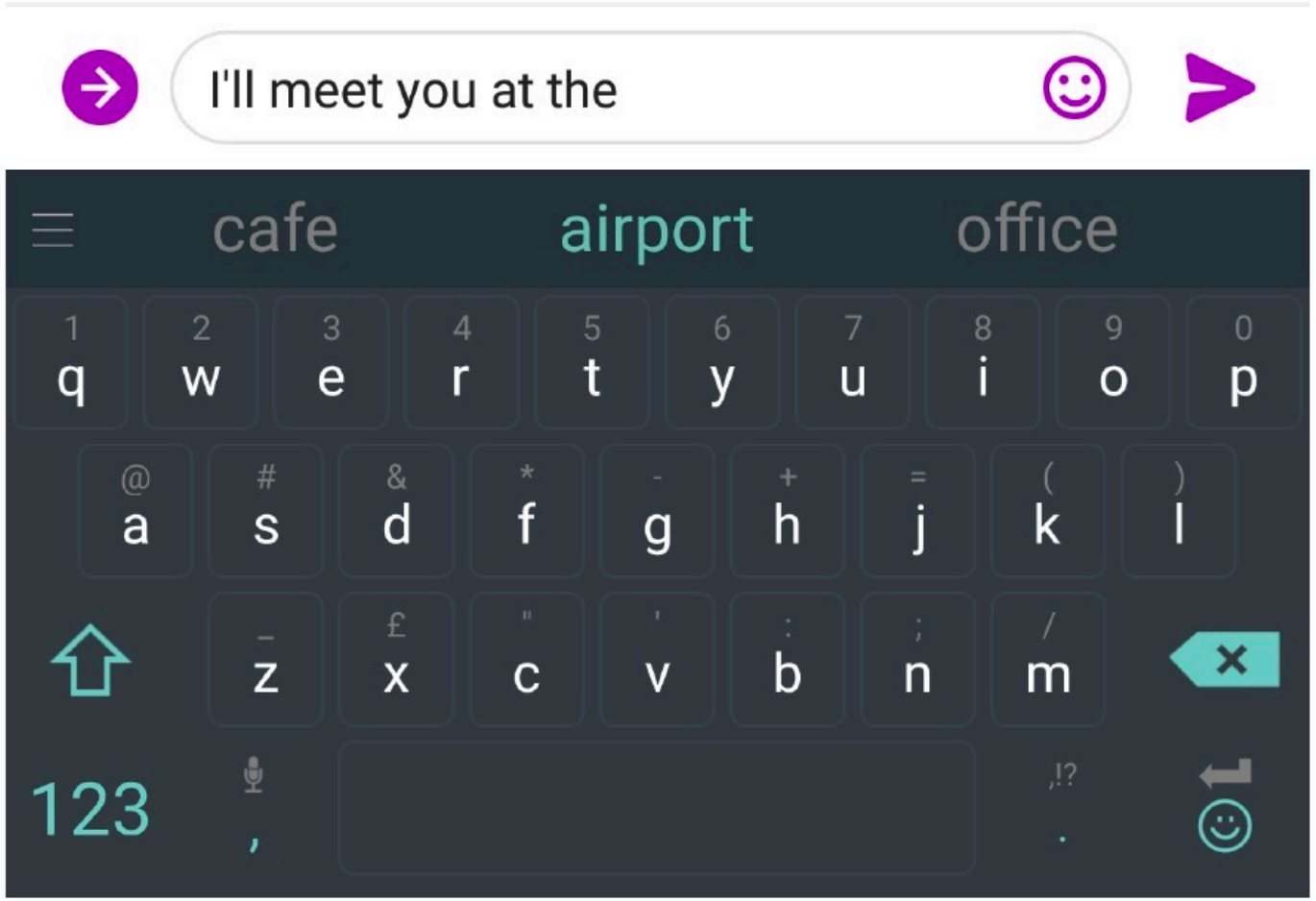
- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

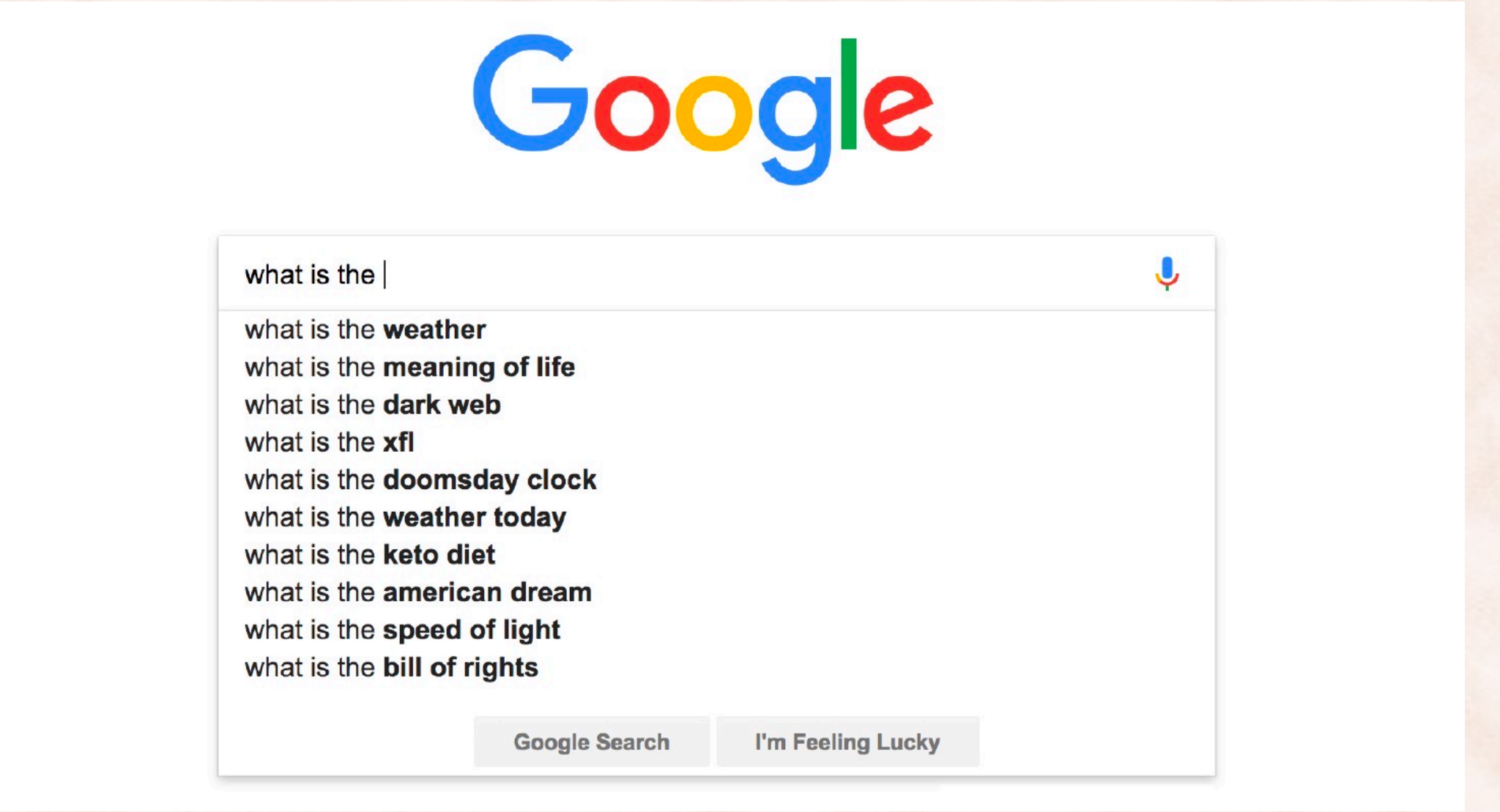
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Everyday Uses of LMs



Everyday Uses of LMs



Outline

- **Language modeling definition**:
  - Causal vs. Masked language modeling.
- **N-gram language models**.
- **Neural language models**.
  - Encoder, Decoder, Encoder-Decoder architectures.
  - FCNs → RNNs → Transformer
- **Large Language Models (LLM) basics**:
  - Autoregressive Generation and Training Scheme.

- **The AI Inverse Problem**:
  - From predicting words to AI.
- **What is Intelligence?**

N-gram Language Models

the students opened their _____

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn a *n*-gram Language Model!

- Definition: A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

N-gram Language Models

- First we make a simplifying assumption: $\boldsymbol{x}^{(t+1)}$ depends only on the preceding *n-1* words.

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

(definition of conditional prob)

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

N-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Sparsity Problems with N-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}$*"* never occurred in data? Then $\boldsymbol{w}$ has probability 0!

(Partial) Solution: Add small δ to the count for every $\boldsymbol{w} \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}$!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse*. Typically we can't have *n* bigger than 5.

Storage Problems with N-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

N-gram Language Models in Practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the _____

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

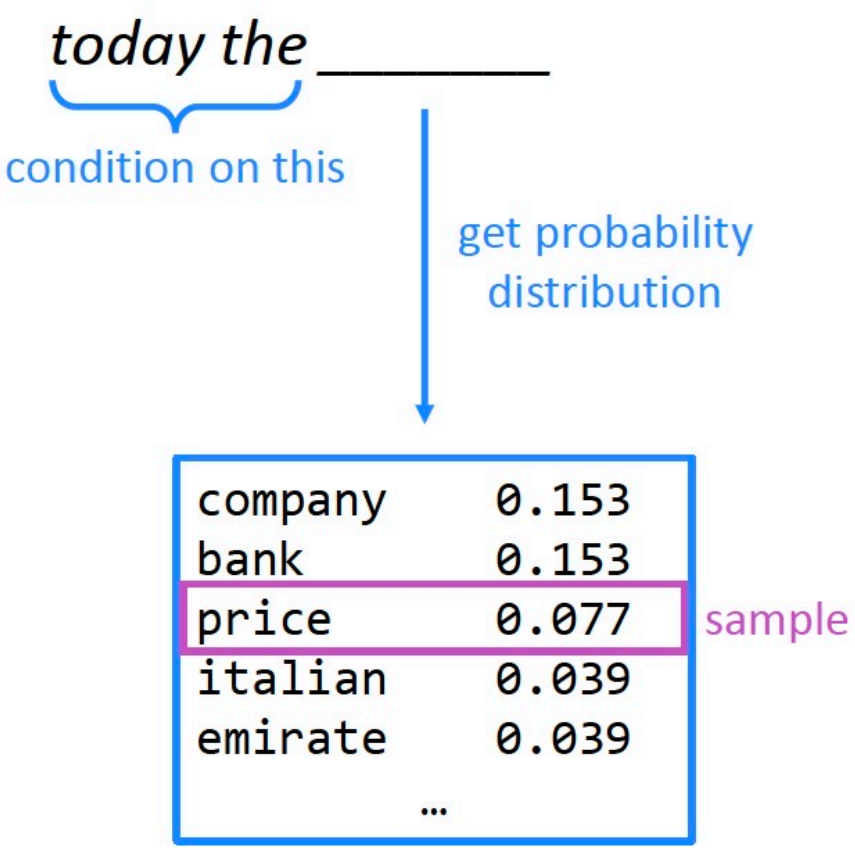
Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

* Try for yourself: https://nlpforhackers.io/language-models/

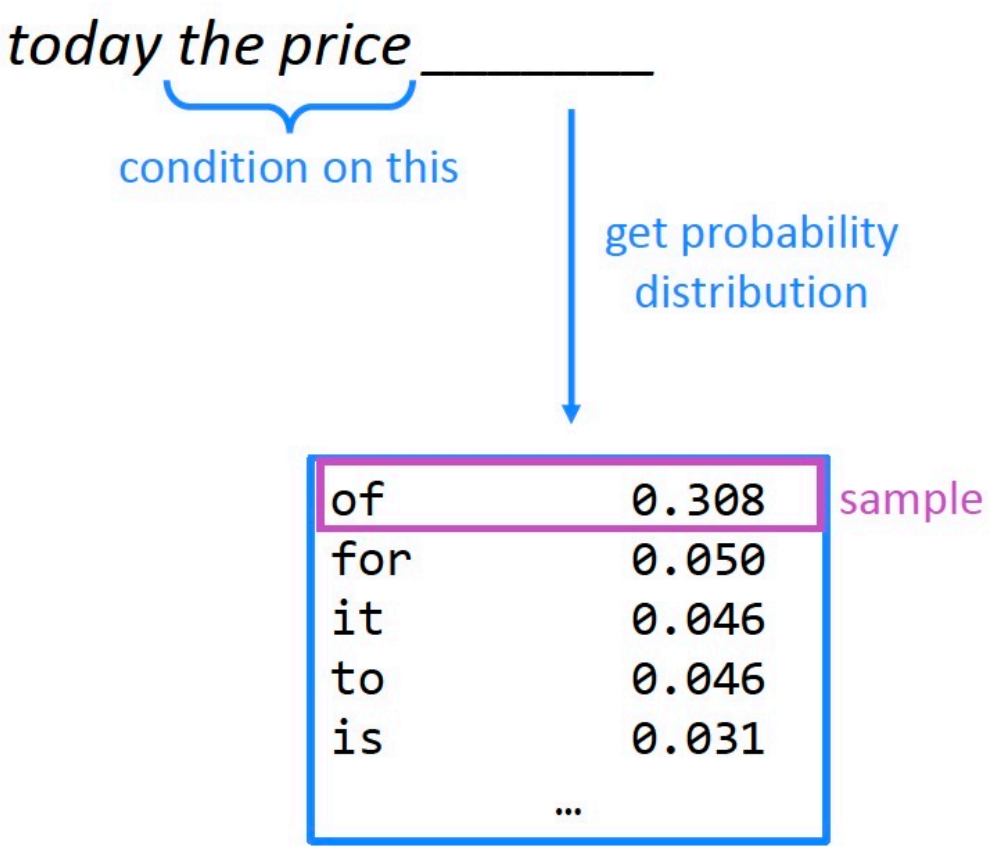
Generating Text with a Language Model

- You can also use a Language Model to generate text.



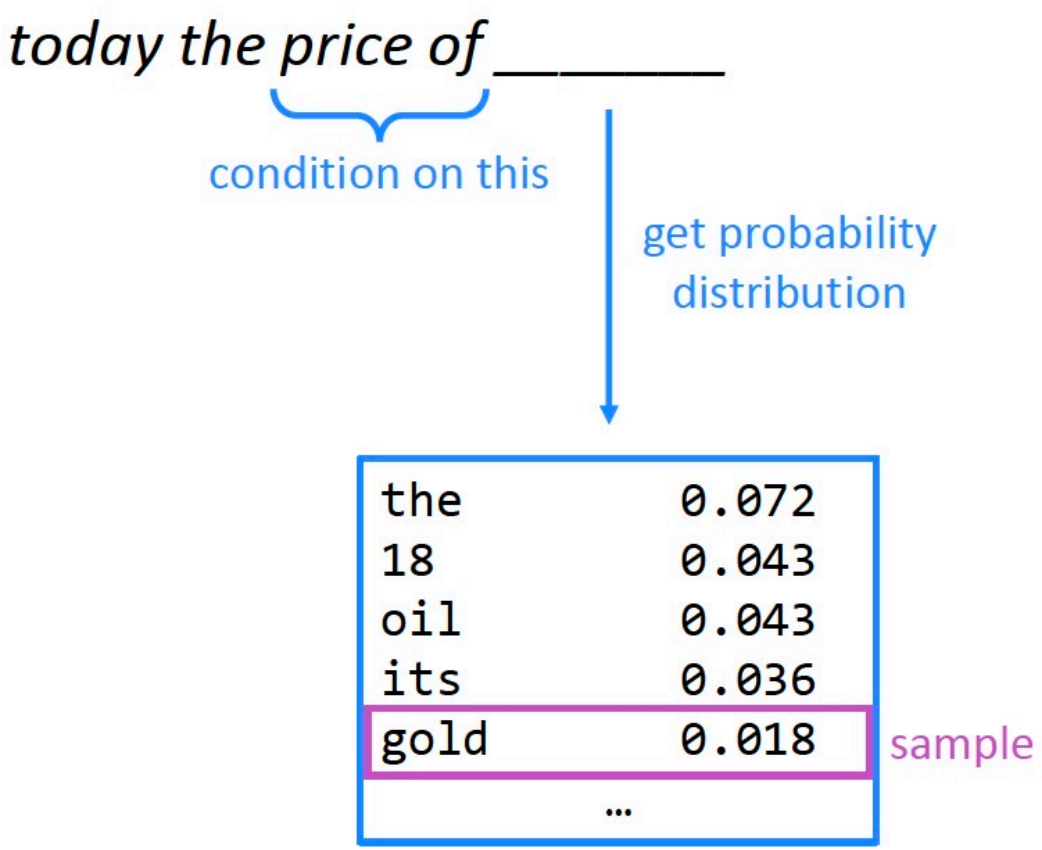
Generating Text with a Language Model

- You can also use a Language Model to generate text.



Generating Text with a Language Model

- You can also use a Language Model to generate text.



Generating Text with a Language Model

- You can also use a Language Model to generate text.

today the price of gold _____

Generating Text with a Language Model

- You can also use a Language Model to generate text.

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size…

N-gram Language Models are Shallow Learners

- N-gram LMs exhibit extremely little "*understanding*" of language due to their **weak generalization**.

- N-gram LMs do not generalize to:
  - Unseen words:
    - *spinach* and *kale* are considered completely different words.
    - sim(*spinach*, *kale*) = sim(*spinach*, *laptop*) = 0.
  - Unseen combinations of known words.
  - Longer context.

Language Models

Evaluating Language Models

Better LMs are better at predicting text

Reminder of the chain rule:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1})$$
$$= \prod_{i=1}^{n} P(w_i|w_{<i})$$

So given a text $w_{1:n}$ we could just compare the log likelihood from two LMs:

$$\log \text{likelihood}(w_{1:n}) = \log \prod_{i=1}^{n} P(w_i|w_{<i})$$

But raw log-likelihood has problems

Probability depends on size of test set:

- Probability gets smaller the longer the text
- We would prefer a metric that is **per-word**, normalized by length.

Perplexity is the inverse probability of the test set, normalized by the number of words:

- The inverse comes from the original definition of perplexity from cross-entropy rate in information theory.
- Probability range is [0,1], perplexity range is [1,∞

Perplexity

We use perplexity to measure how well the LM predicts unseen text.

The perplexity of a model θ on an unseen test set is the **inverse probability that θ assigns to the test set, normalized by the test set length.**

For a test set of *n* tokens $w_{1:n}$ the perplexity is :

$$\text{Perplexity}_\theta(w_{1:n}) = P_\theta(w_{1:n})^{-\frac{1}{n}} = \sqrt[n]{\frac{1}{P_\theta(w_{1:n})}} = \sqrt[n]{\prod_{i=1}^{n}\frac{1}{P_\theta(w_i|w_{<i})}}$$

Perplexity

- The higher the probability of the word sequence, the lower the perplexity.
- Thus, the lower the perplexity of a model on the data, the better the model.
- **Minimizing perplexity is the same as maximizing probability**.

Also: perplexity is sensitive to length/tokenization so best used when comparing LMs that use the same tokenizer.

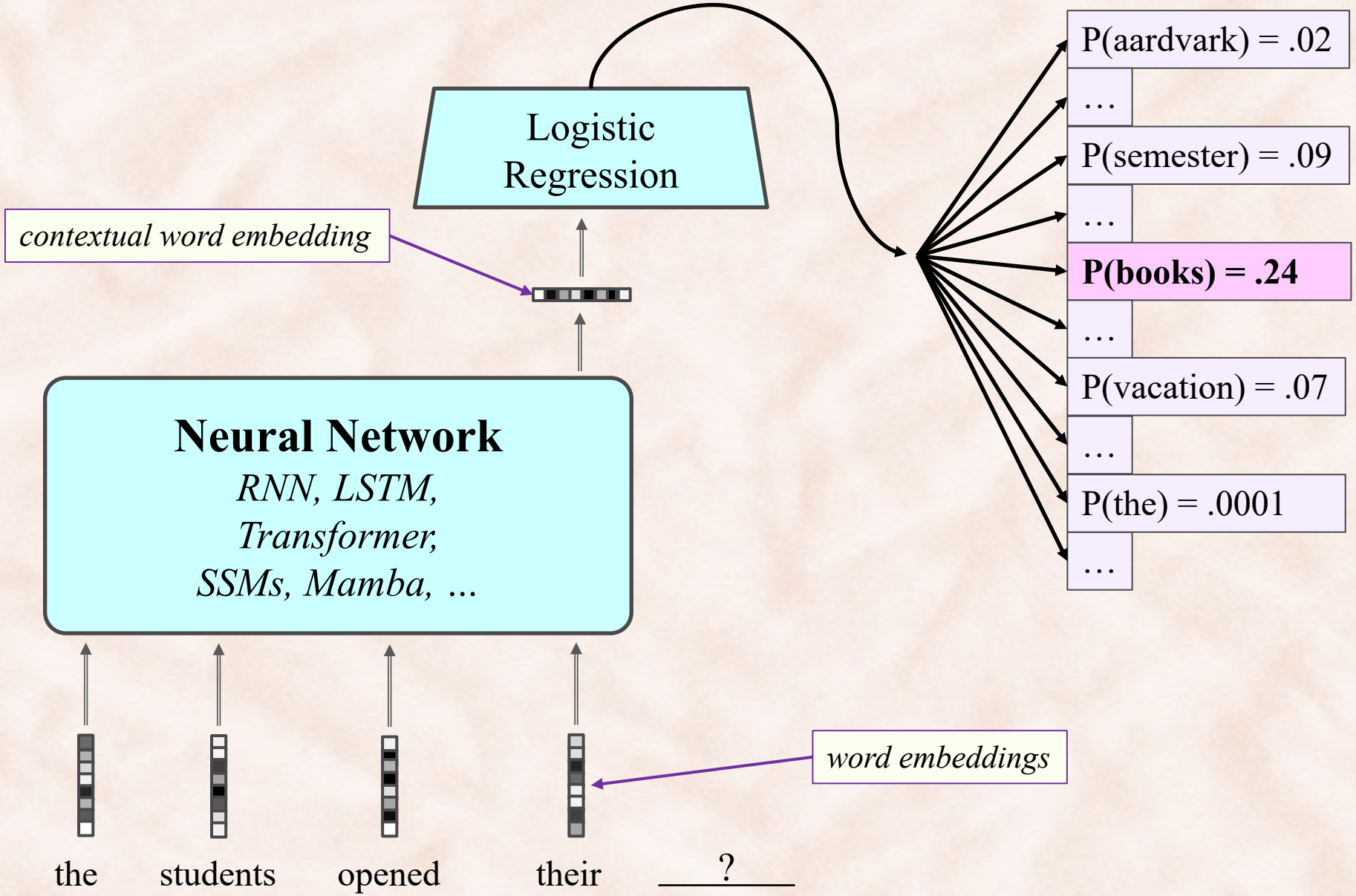
Outline

- **Language modeling definition**:
  - Causal vs. Masked language modeling.
- **N-gram language models**.
- **Neural language models**.
  - Encoder, Decoder, Encoder-Decoder architectures.
  - FCNs → RNNs → Transformer
- **Large Language Models (LLM) basics**:
  - Autoregressive Generation and Training Scheme.

- **The AI Inverse Problem**:
  - From predicting words to AI.
- **What is Intelligence?**

Neural LMs

https://www.sciencefocus.com/planet-earth/asteroid-impact-change-earths-orbit


Logistic Regression
contextual word embedding
Neural Network
RNN, LSTM,
Transformer,
SSMs, Mamba, ...
P(aardvark) = .02
…
P(semester) = .09
…
P(books) = .24
…
P(vacation) = .07
…
P(the) = .0001
…
word embeddings
the
students
opened
their
___?___

Word Embeddings

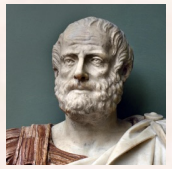
- Neural LMs use as input **word embeddings**:
  - **Vectors** that are **short**, e.g. 256, 512, 768, 1024, or 3,072, and **dense**, e.g. most if not all entries are non-zero.
    - Very unlike *one-hot encodings,* which are long and sparse.

- We want word embeddings to represent **word meanings**:
  - What do we mean by "word meaning"?
  - How do we train word embeddings to represent word meaning?
    - How do we determine their quality?

What do words mean?

- Classical approach uses a **dictionary** and the **context**.

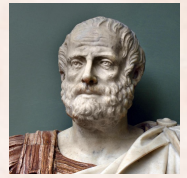
- Depending on the context, a word is used to refer to a particular *concept*, i.e. *sense*, i.e. *word meaning*:
  - The word "pepper" has multiple meanings, as listed in the dictionary:
    - **sense 1**: spice from pepper plant
    - **sense 2**: the pepper plant itself
    - **sense 3**: another similar plant (Jamaican pepper)
    - **sense 4**: another plant with peppercorns (California pepper)
    - **sense 5**: capsicum (i.e. chili, paprika, bell pepper, etc)



Aristotle: What is a concept?

- **Classical theory** of concepts:
  - The meaning of a word is a concept that is defined by **necessary** and **sufficient** conditions.
    - The following necessary conditions, jointly, are sufficient for an object *x* to be a **square**:
      - *x* has (exactly) four sides
      - each of *x*'s sides is straight
      - *x* is a closed figure
      - *x* lies in a plane
      - each of *x*'s sides is equal in length to each of the others
      - each of *x*'s interior angles is equal to the others (right angles)
      - the sides of *x* are joined at their ends

Wittgenstein to Aristotle: *But what is a **game**?*



- Philosophical Investigations (1945, # 66):

  Don't say "*there must be something common, or they would not be called 'games'*"—but *look and see* whether there is anything common to all"

  – Is it amusing? Is there competition? Is there long-term strategy?

  – Is skill required? Must luck play a role?

  – Are there cards? Is there a ball?

  – …

Distributional Semantics Idea

- Wittgenstein (1945):

  The meaning of a word is its use in the language.

- Harris (1954):

  Words that occur in the same contexts tend to have similar meanings.

- Firth (1935, 1957):

  The complete meaning of a word is always contextual, and no study of meaning apart from a complete context can be taken seriously.

  You shall know a word by the company it keeps.

Distributional Semantics

- The **meaning** of a word is **determined by** the words that appear nearby, i.e. its **context**.
  - **Words that appear in the same contexts tend to have similar meanings.**
  - One of the most successful ideas of modern statistical NLP!

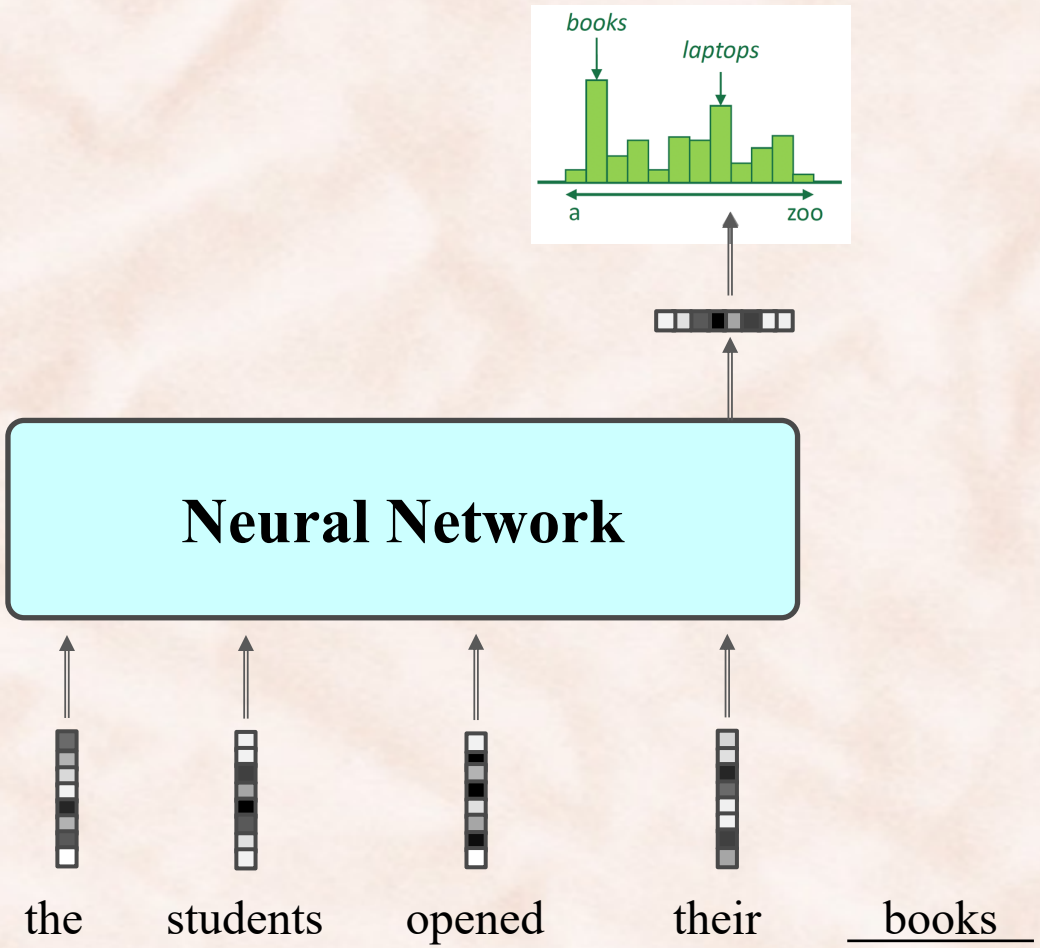
Distributional Semantics and Neural LMs

- Learn word embeddings by training a neural network to **predict a missing word** given words in the **context**.
  - This is a special type of *reconstructing the input* idea used in other modalities, such as computer vision (see *autoencoders*).

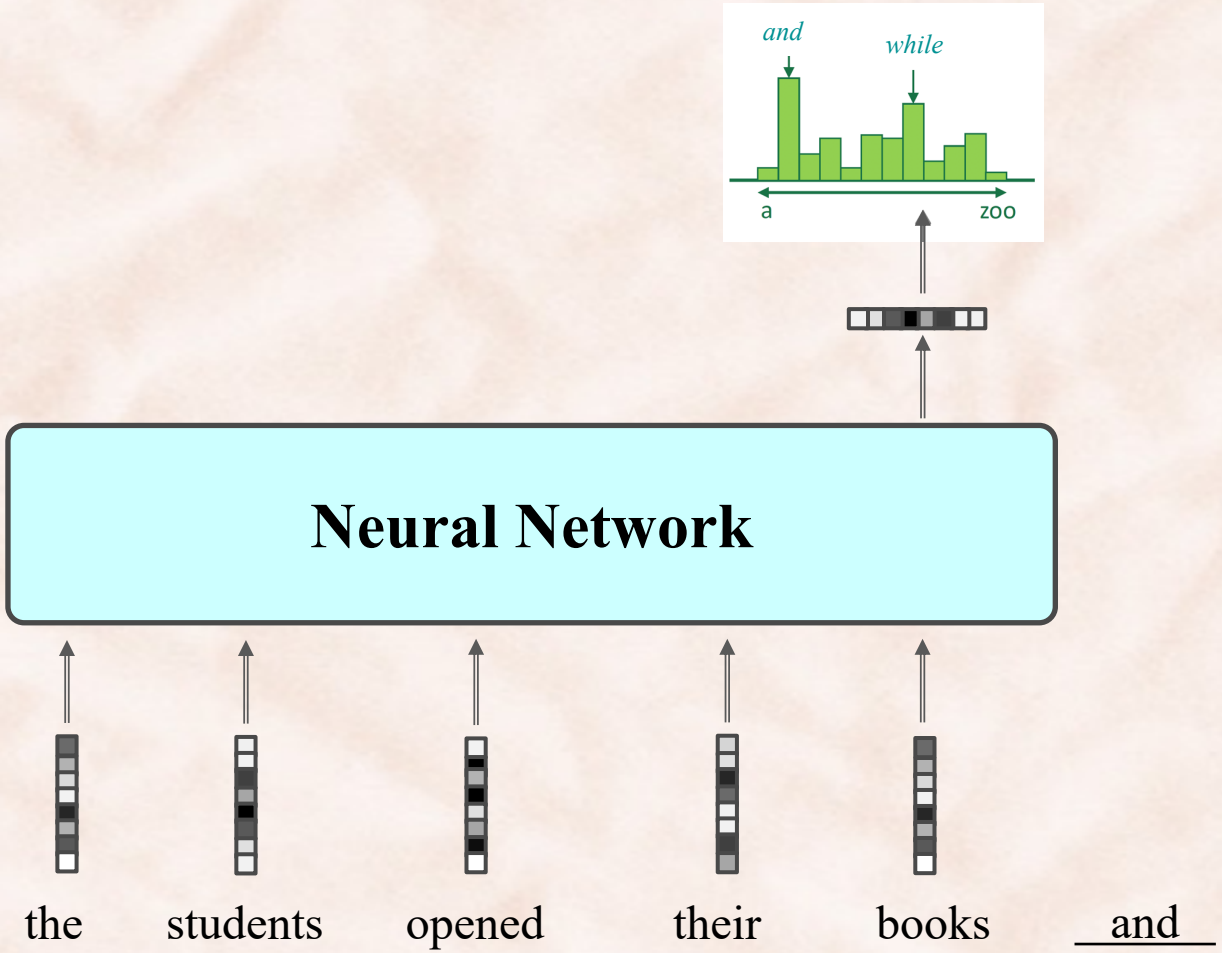
- **Causal LMs** trained using the distributional hypothesis:
  - Context: words so far.
  - Missing word: next word.

- **Masked LMs** trained using the distributional hypothesis:
  - Context: words to the left and to the right of the center word.
  - Missing word: word in the center.

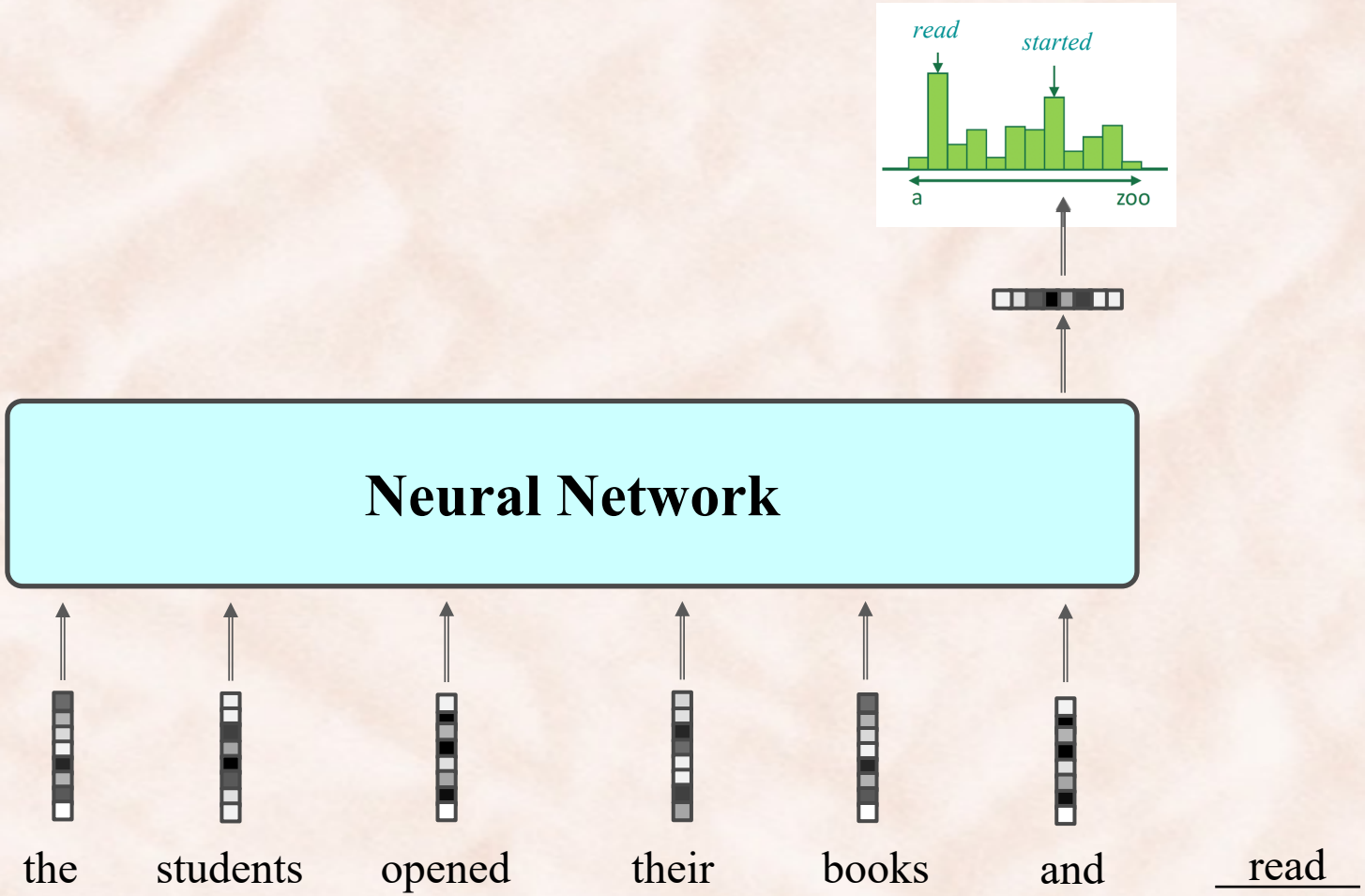
Neural Language Modeling: **Decoder**



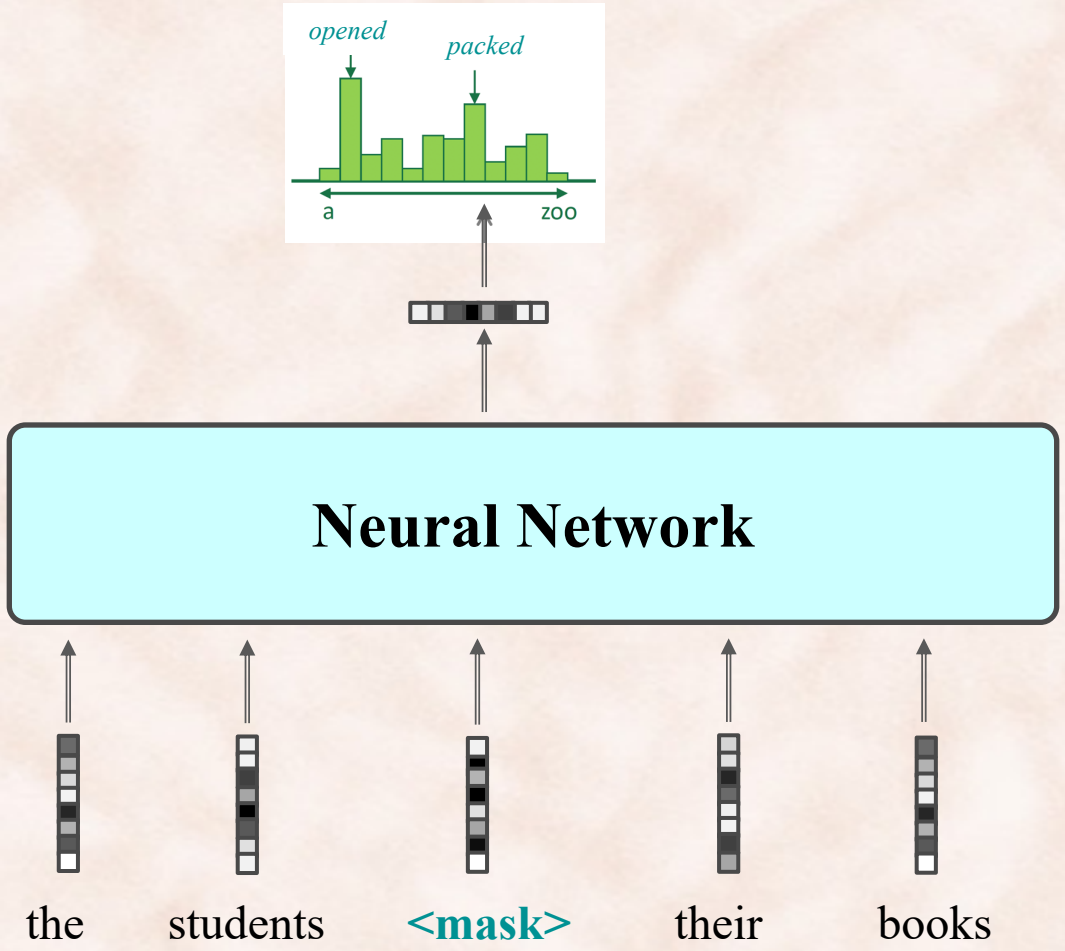
Neural Language Modeling: **Decoder**



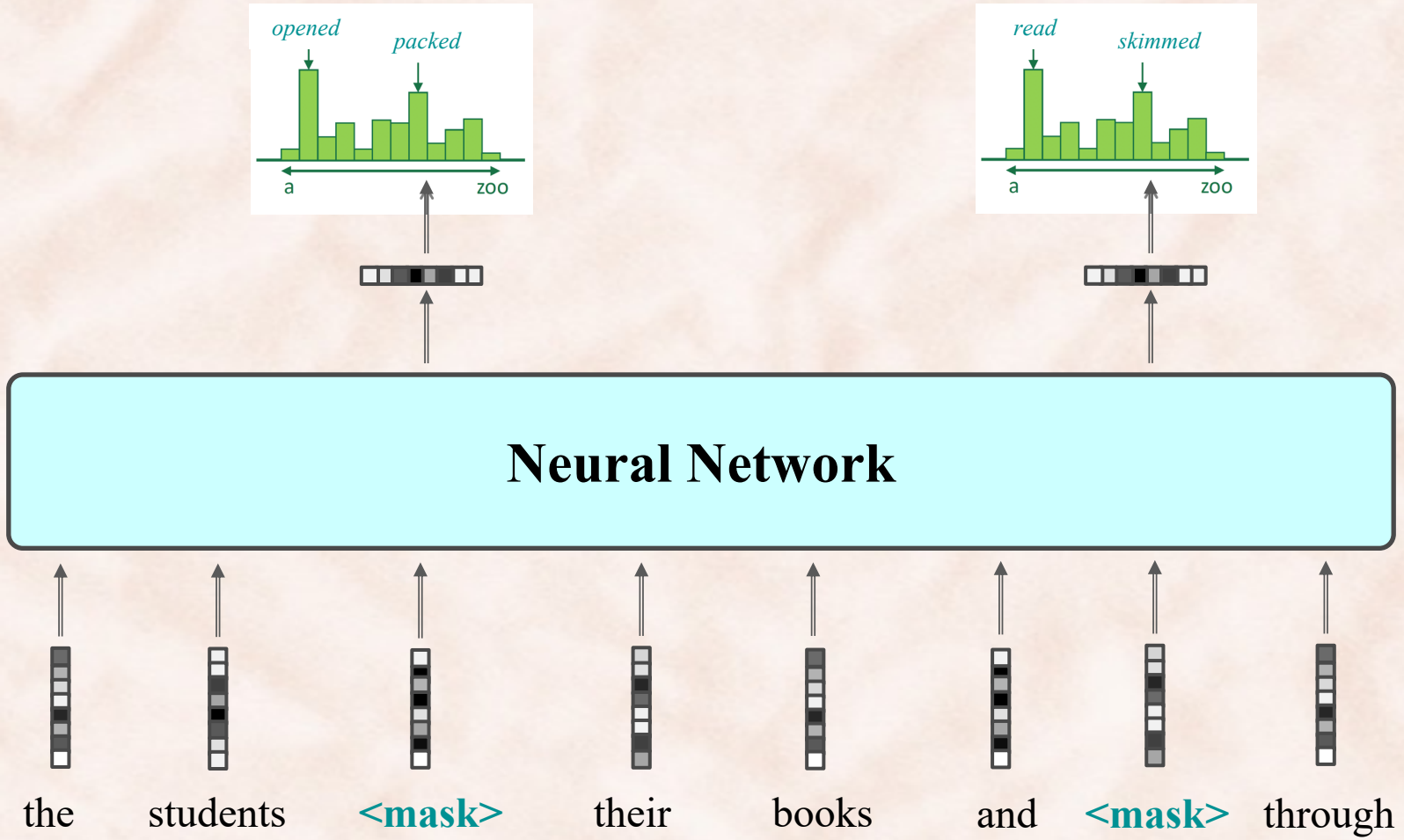
Neural Language Modeling: **Decoder**



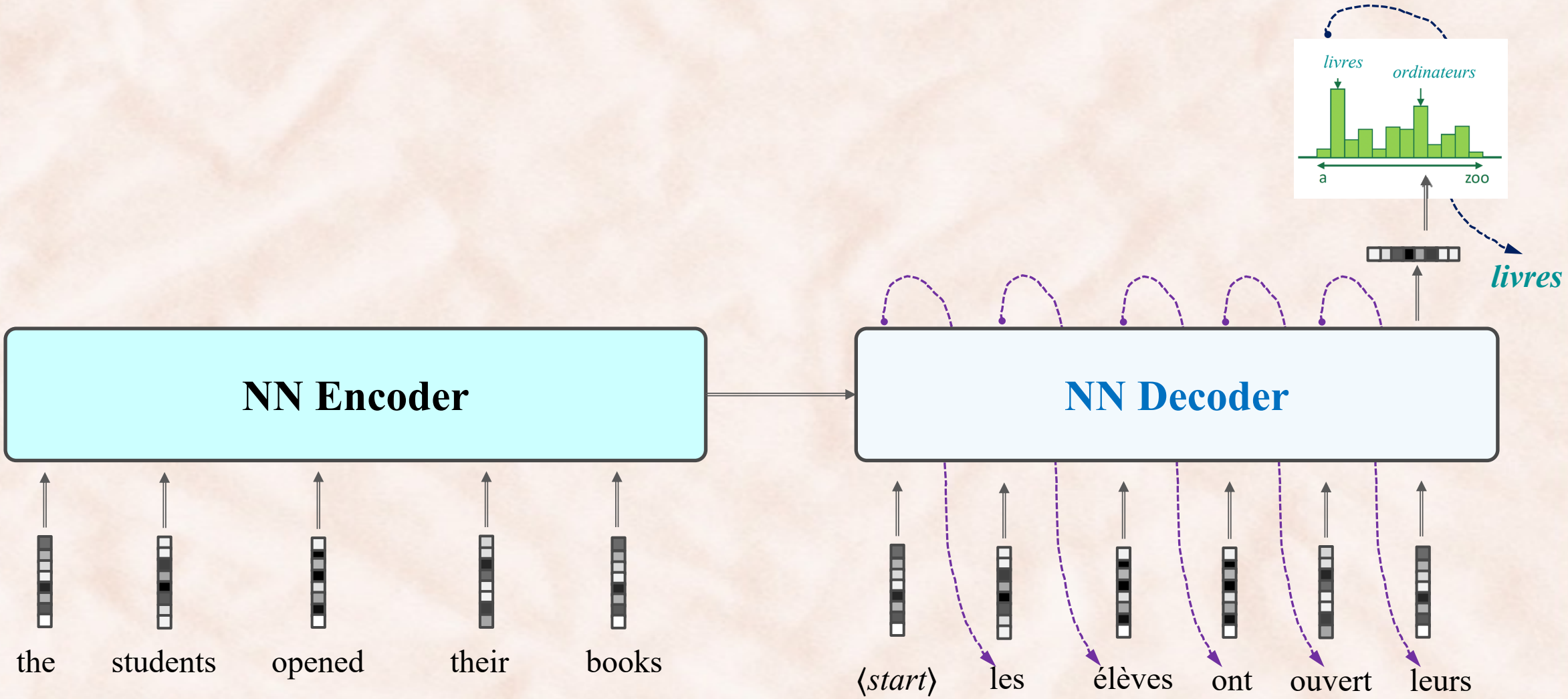
Neural Language Modeling: **Encoder**



Neural Language Modeling: **Encoder**



Neural Language Modeling: **Encoder-Decoder**



Outline

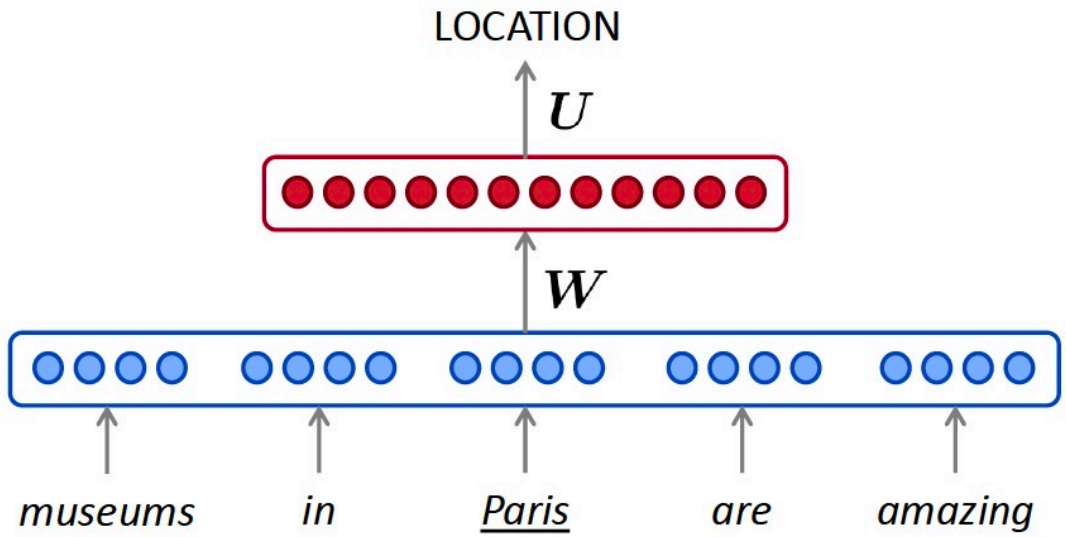
- **Language modeling definition**:
  - Causal vs. Masked language modeling.
- **N-gram language models**.
- **Neural language models**.
  - Encoder, Decoder, Encoder-Decoder architectures.
  - FCNs → RNNs → Transformer
- **Large Language Models (LLM) basics**:
  - Autoregressive Generation and Training Scheme.

- **The AI Inverse Problem**:
  - From predicting words to AI.
- **What is Intelligence?**

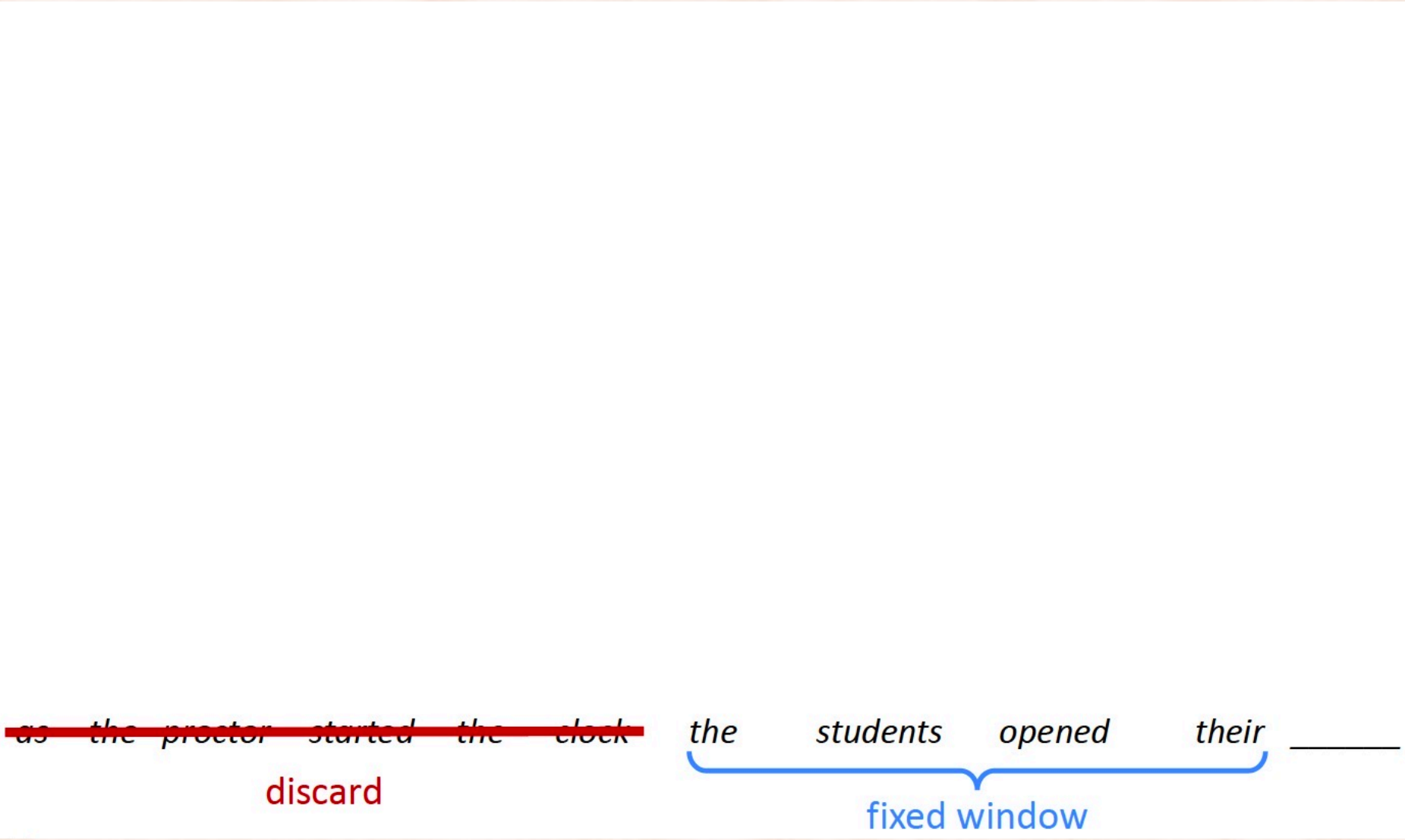
How to build a Neural LM?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob dist of the next word $P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?
  - We can apply this to Named Entity Recognition:



Fully Connected Networks (FCNs)



FCNs with a Fixed-Window

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

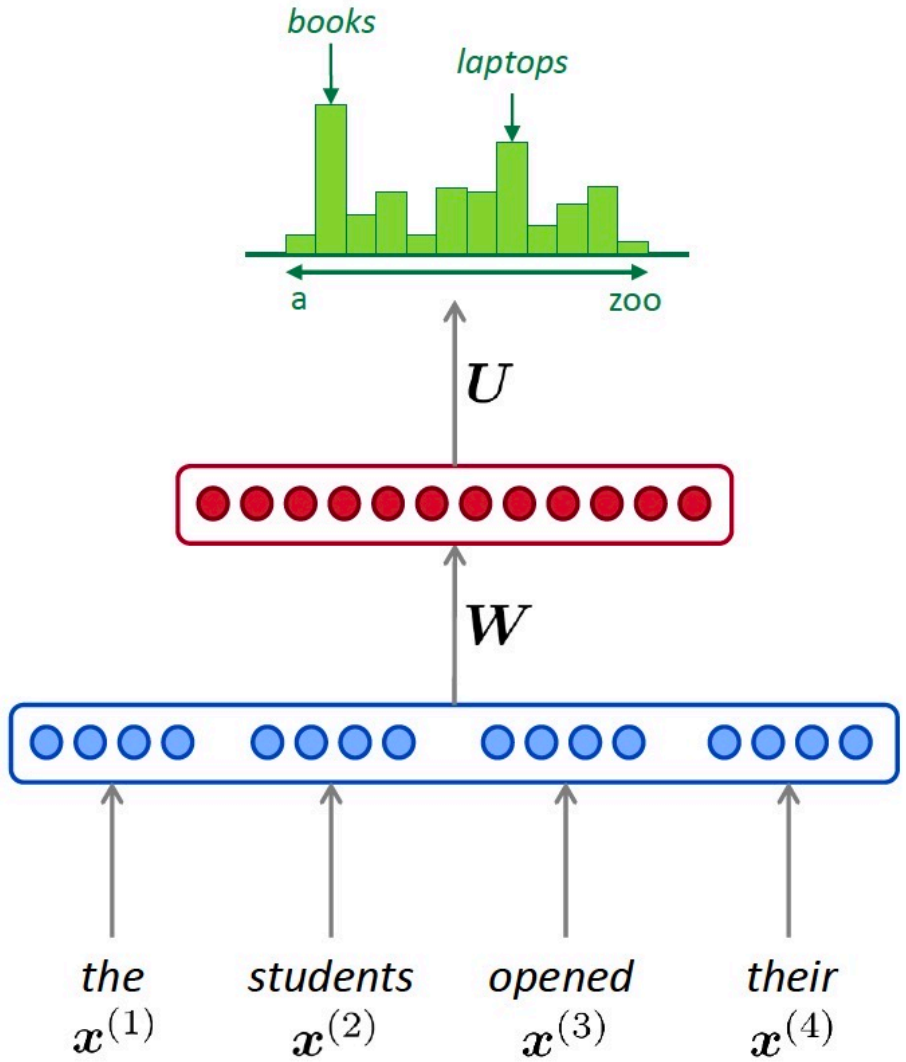
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



FCNs with a Fixed-Window

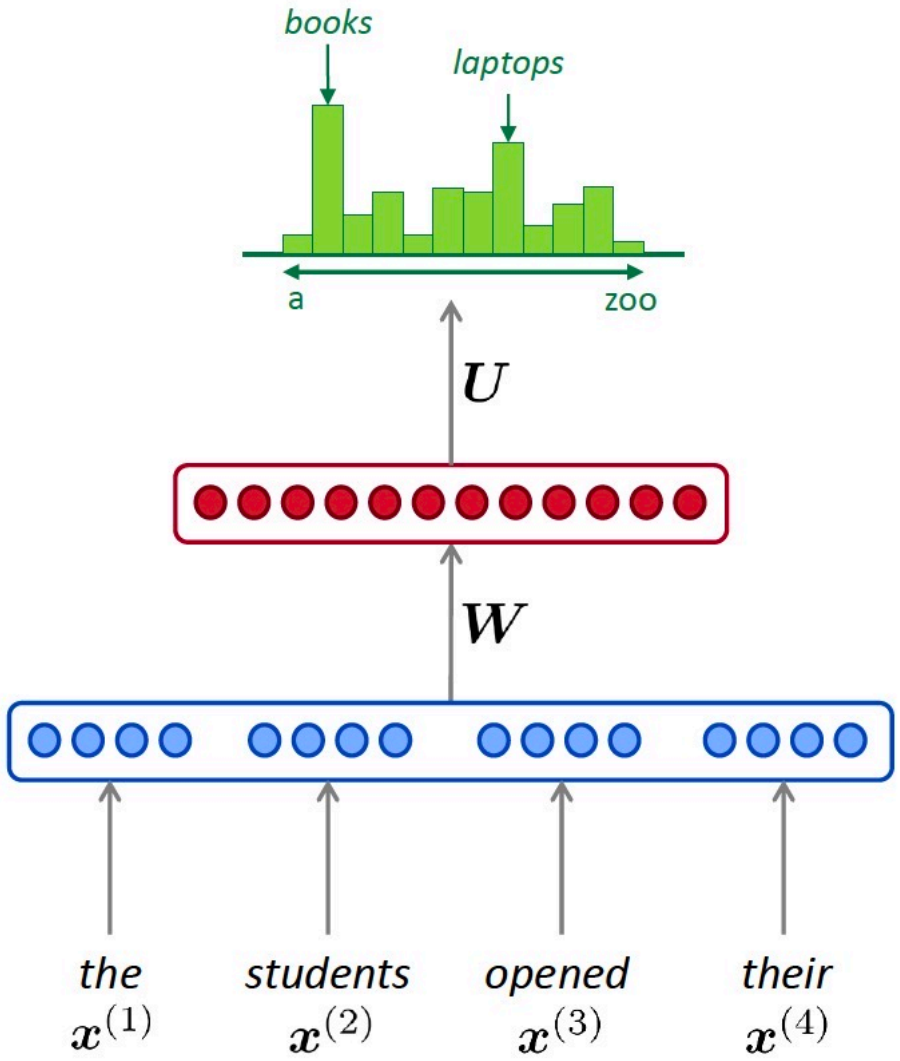
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



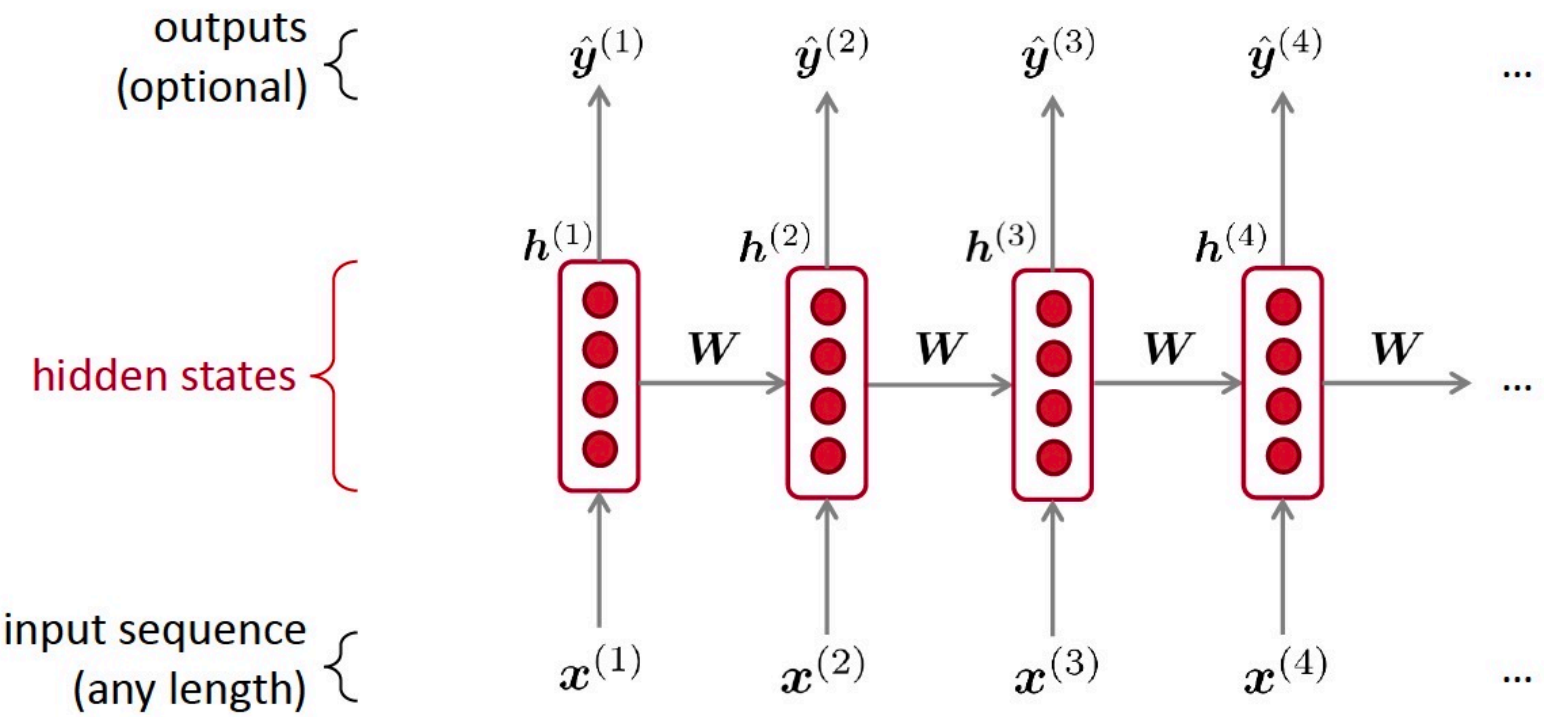
Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W *repeatedly*

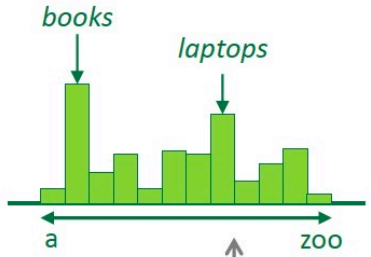
Slides from the CS224N at Stanford



RNN Decoder with Variable-Length Window

A RNN Language Model

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

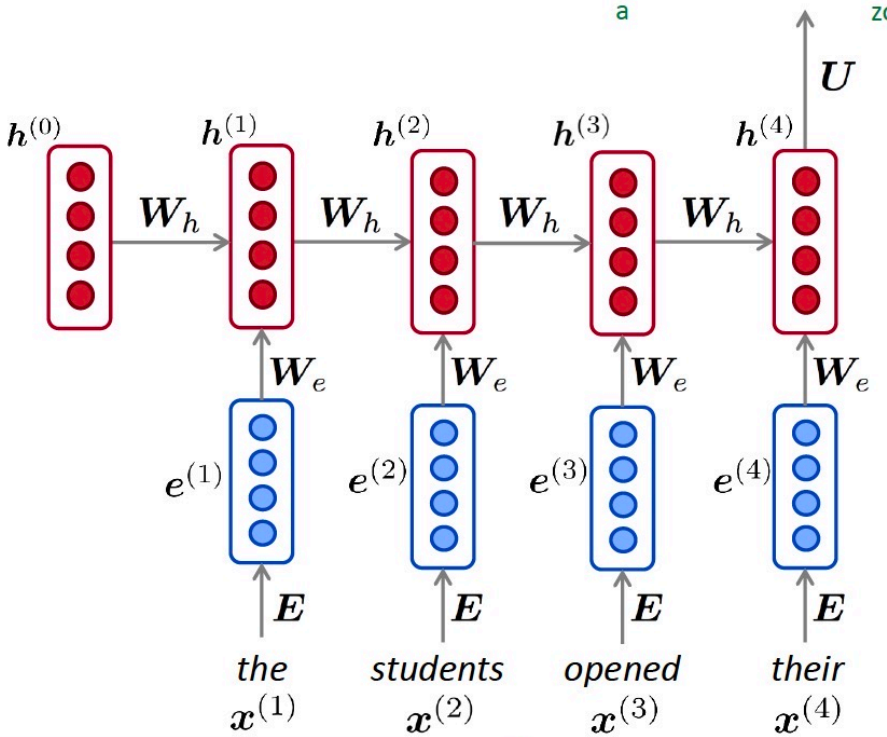
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Note: *this input sequence could be much longer, but this slide doesn't have space!*

RNN Decoder with Variable-Length Window

A RNN Language Model

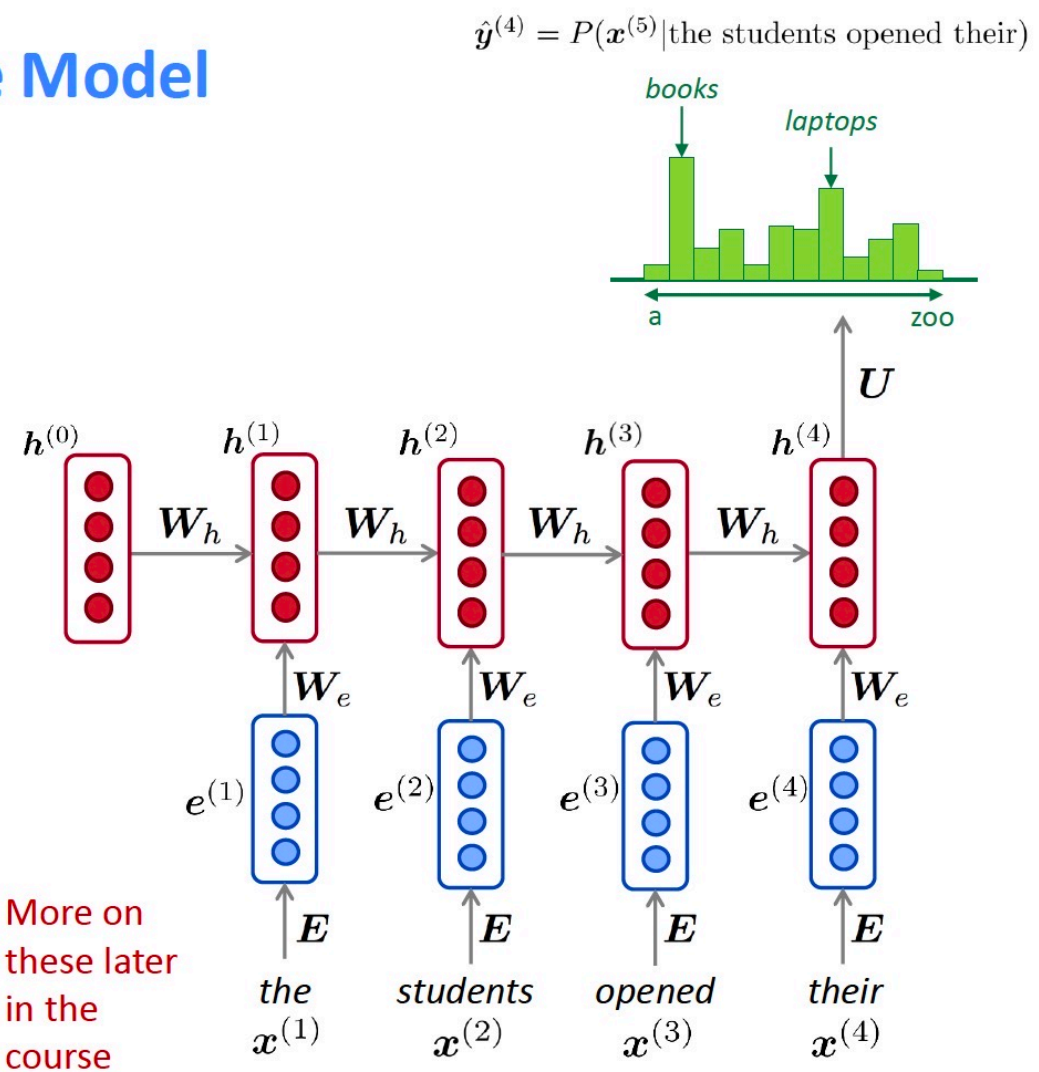
RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

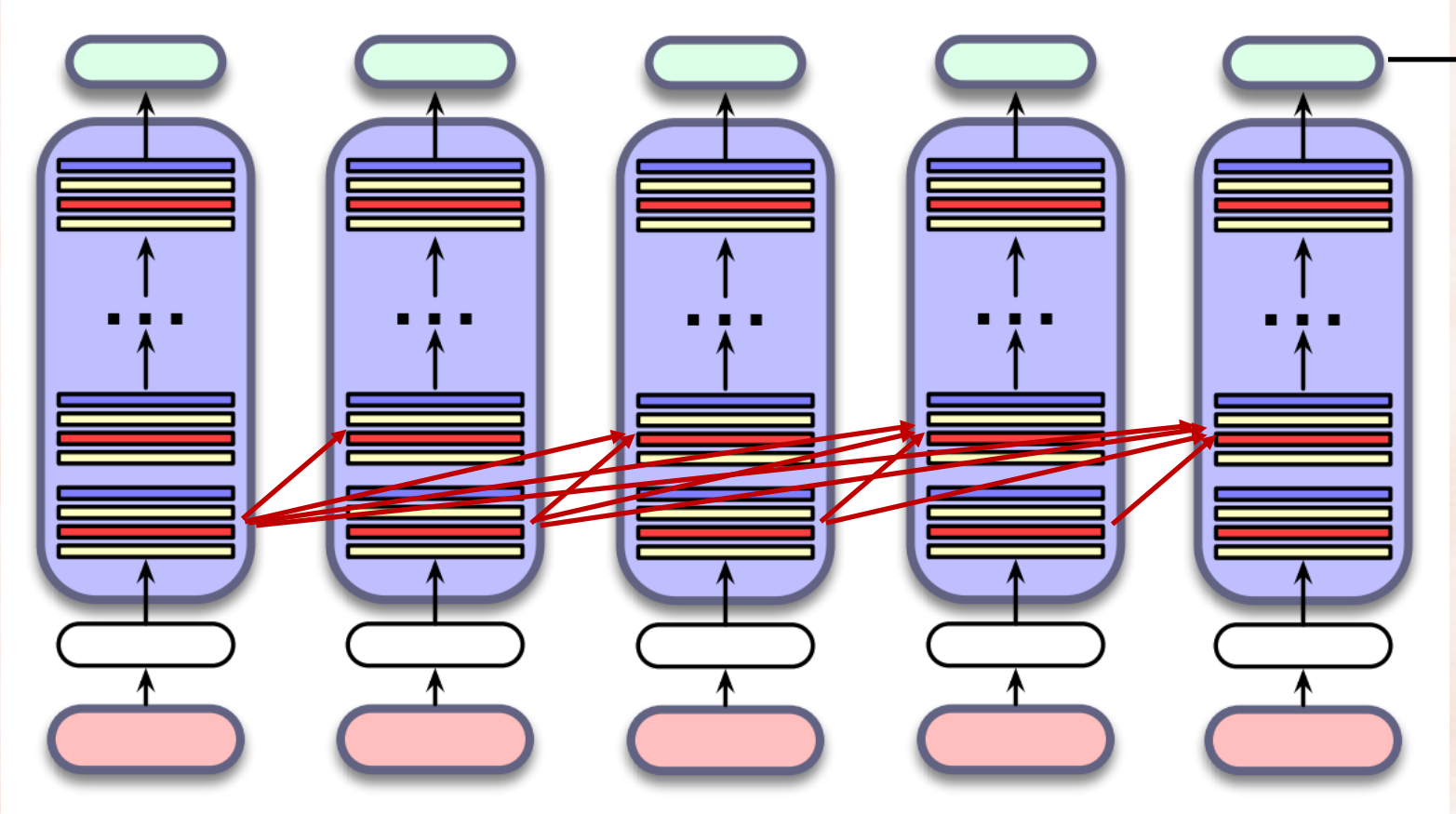
RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course

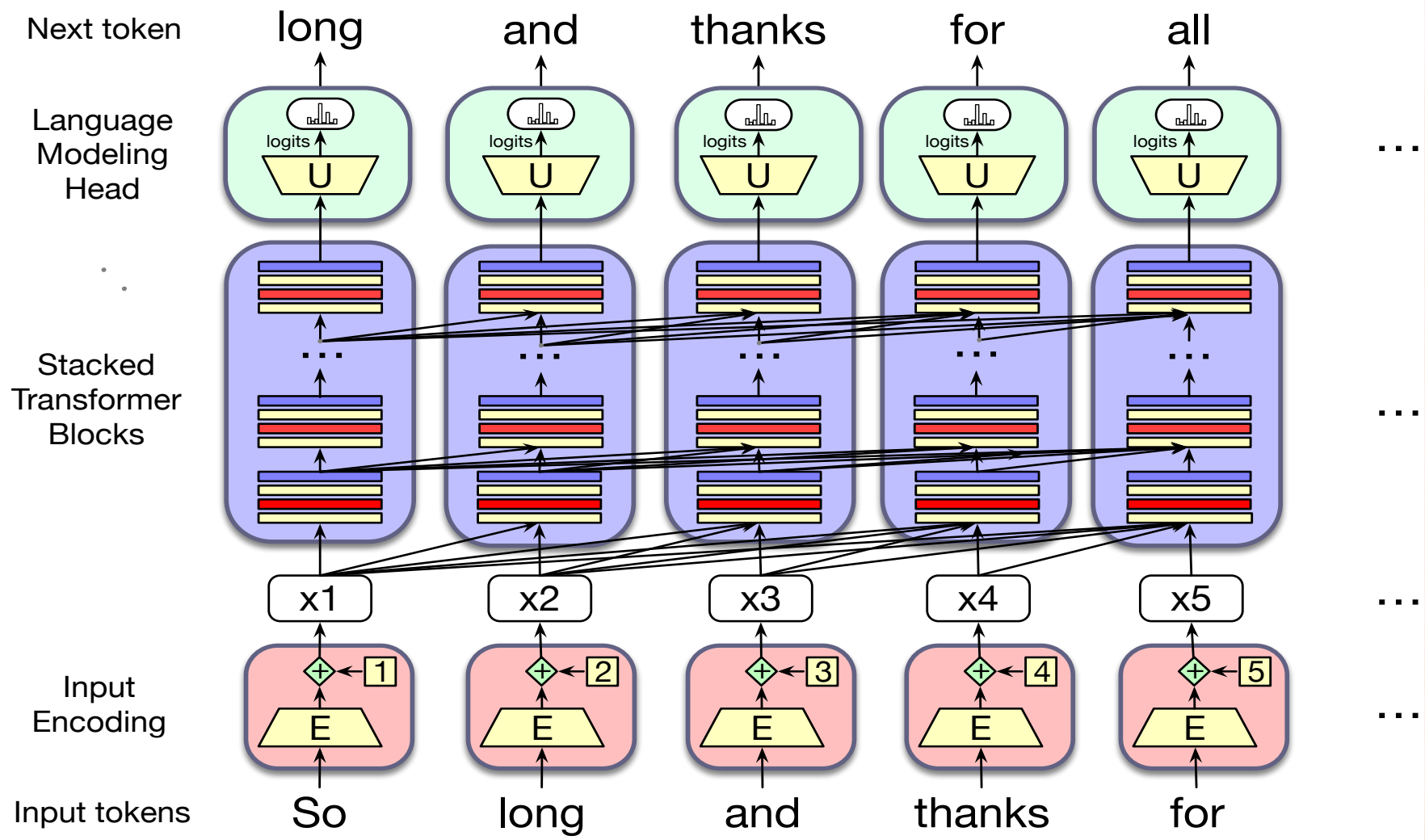


Transformer



- Same parameters are used at each input token position:
  - **Embeddings** & **Softmax**.
  - **FCNs**.
  - **Layer normalization**.
  - **Attention**.

Transformer Decoder with Variable-length Window

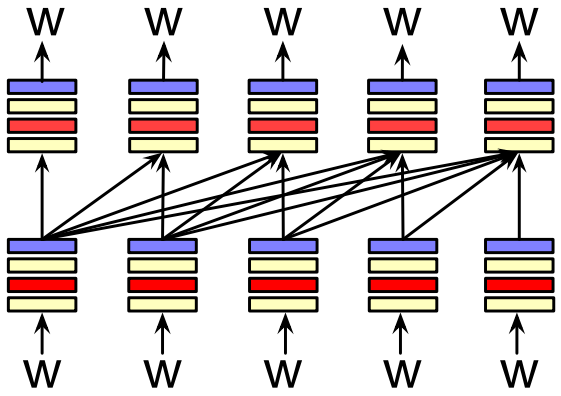


Outline

- **Language modeling definition**:
  - Causal vs. Masked language modeling.
- **N-gram language models**.
- **Neural language models**.
  - Encoder, Decoder, Encoder-Decoder architectures.
  - FCNs → RNNs → Transformer
- **Large Language Models (LLM) basics**:
  - Autoregressive Generation and Training Scheme.

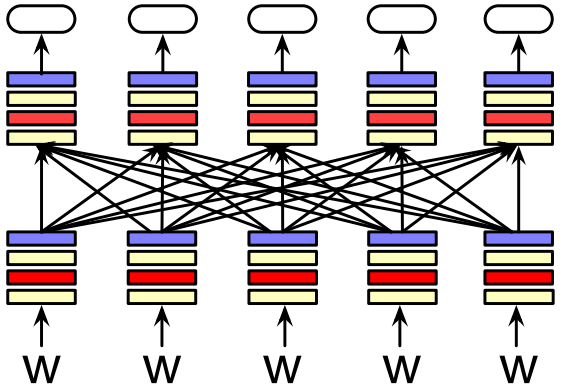
- **The AI Inverse Problem**:
  - From predicting words to AI.
- **What is Intelligence?**

Three architectures for Large Language models



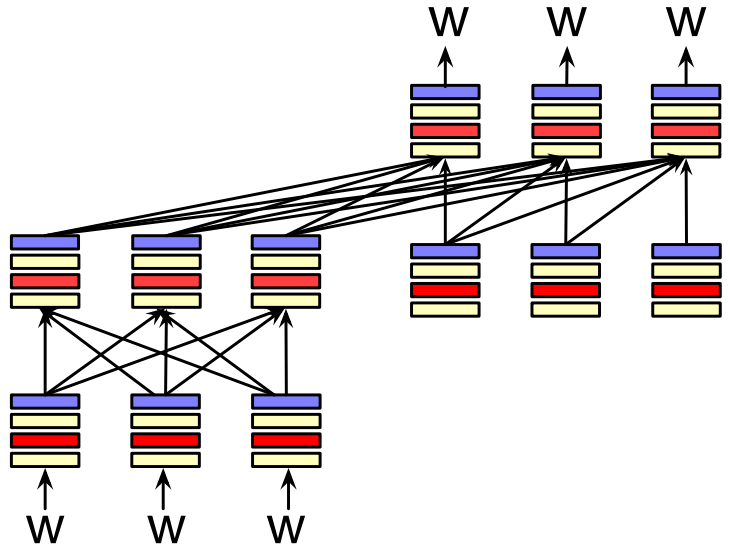
<u>Decoders</u>

GPT, Claude,
Llama, Qwen
Grok



Encoders

BERT family,
RoBERTa,
HuBERT



Encoder-decoders

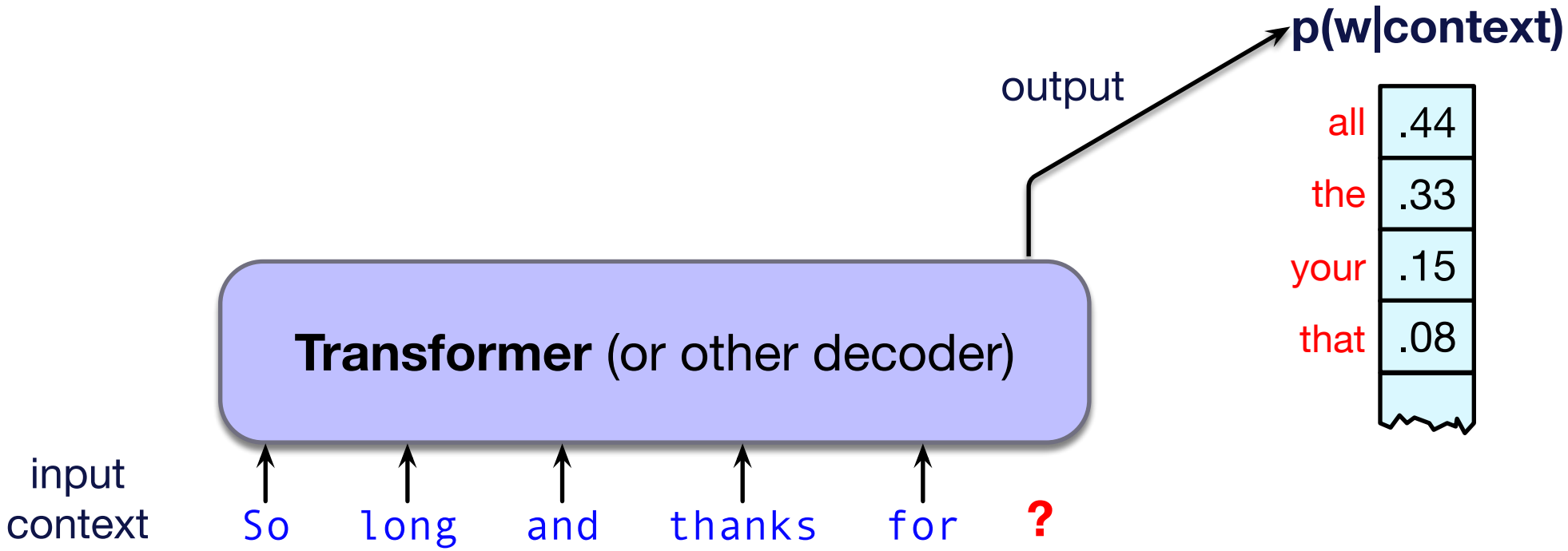
Flan-T5,
Whisper

What is a large language model?

A neural network with:

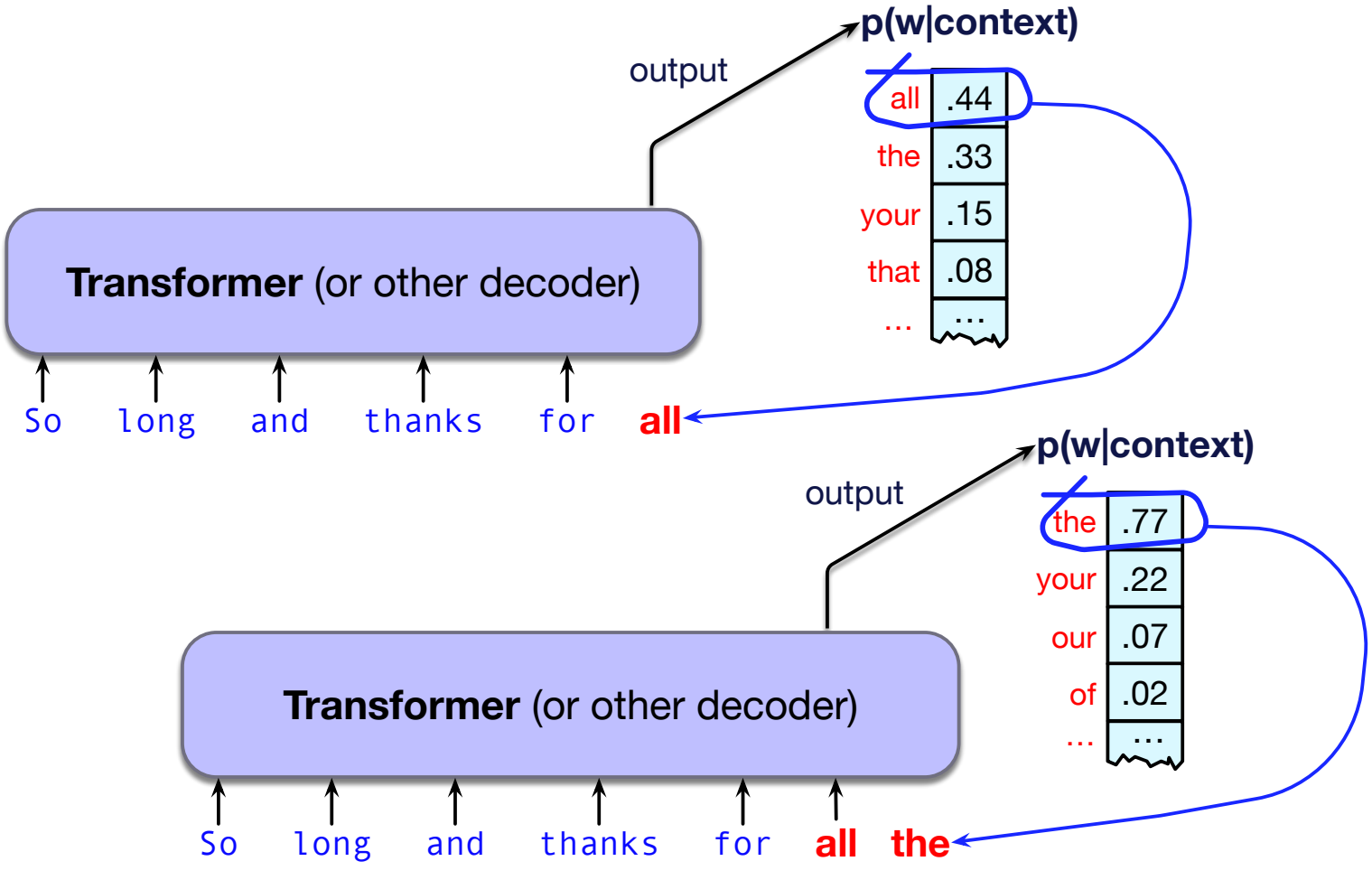
Input: a context or prefix,

Output: a distribution over possible next words



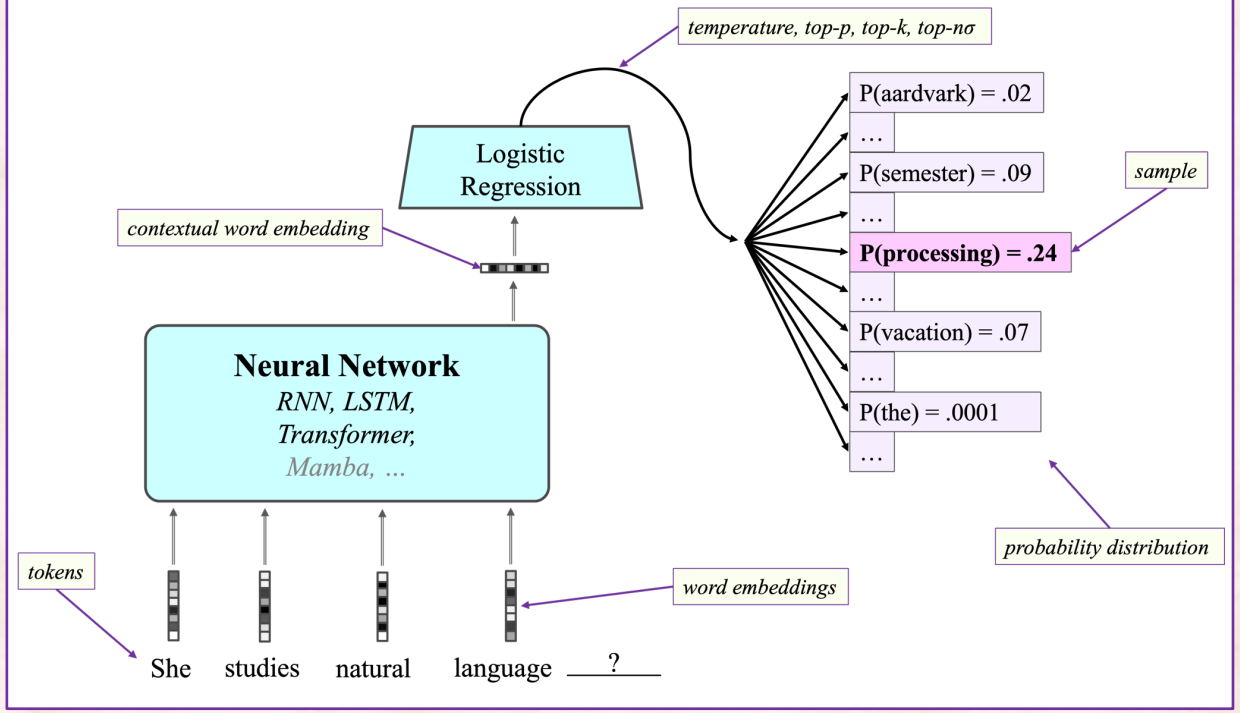
LLMs can generate!

A model that gives a probability distribution over next words can generate by repeatedly sampling from the distribution

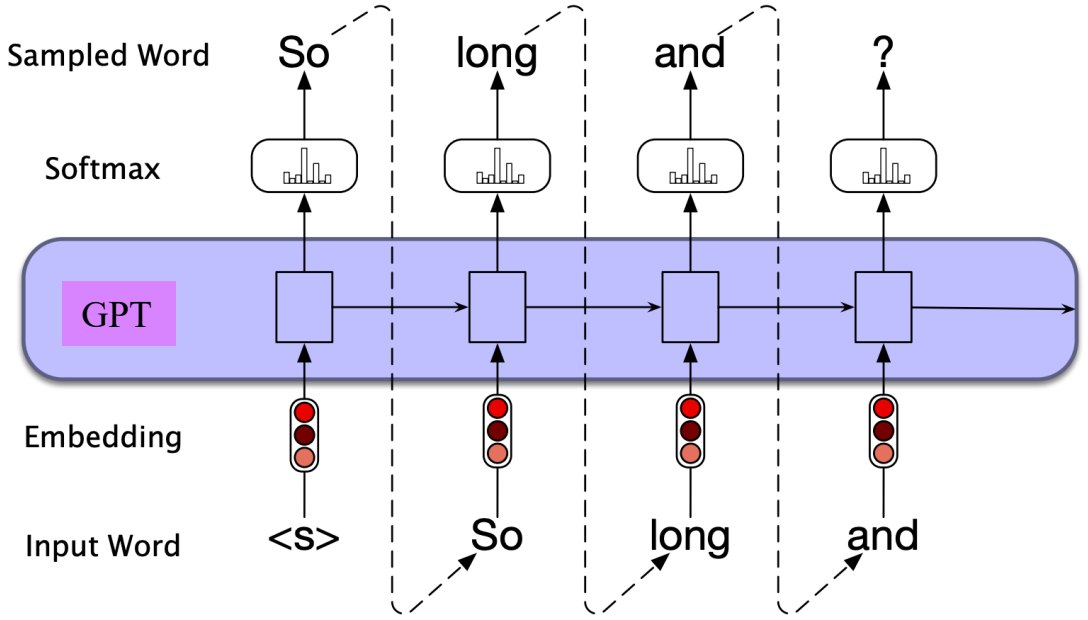


Using LLMs: Autoregressive Generation

LLM **one-token** distribution computation



LLM **multi-token** autoregressive generation



https://web.stanford.edu/~jurafsky/slp3/9.pdf

Large Language Models

Conditional Generation of Text

Autoregressive Conditional Generation

1. Give the LLM an input piece of text, a **prompt**
2. Have it generate text, token by token:
   - conditioned on the prompt and the generated tokens

We generate from a model by

1. computing the probability of the next token w_i from the prior context: $P(w_i | w_{<i})$
2. sampling from that distribution to generate a token

Many NLP tasks can be cast as conditional generation

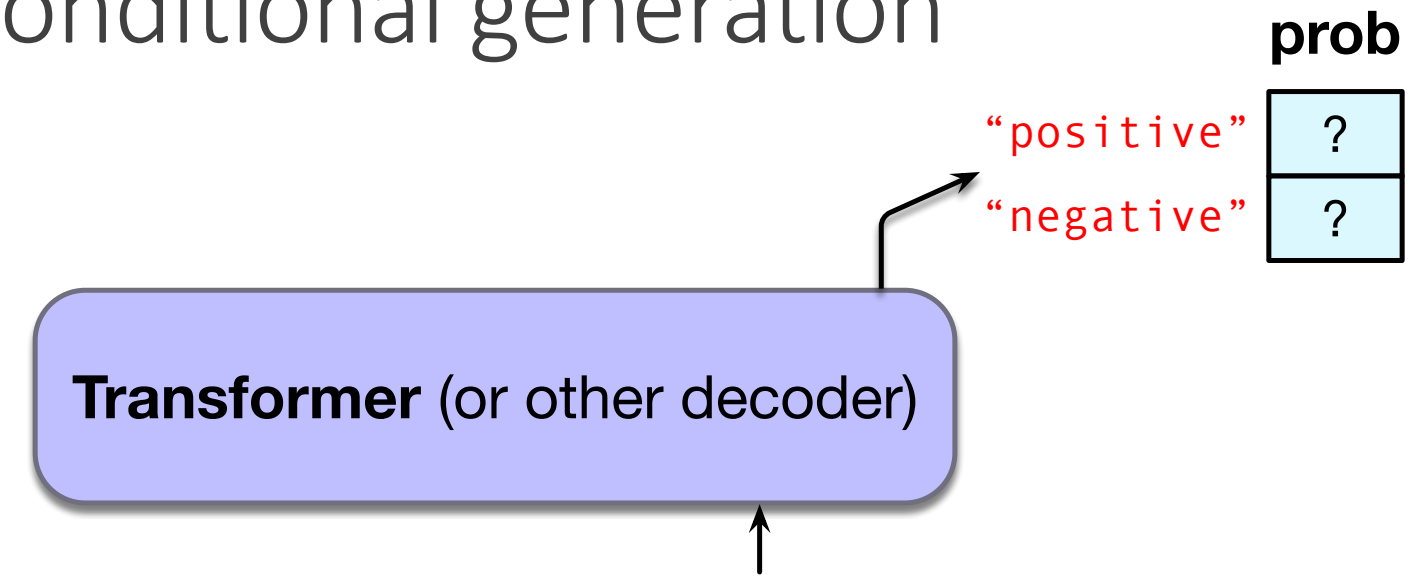
Sentiment analysis: “I like Jackie Chan”

1. We give the language model this string:

   ```
   The sentiment of the sentence "I like Jackie Chan" is:
   ```

2. And see the distribution of the next word:

Sentiment via conditional generation



Which word has a higher probability?

$$P(\text{positive}|\texttt{The sentiment of the sentence ‘‘I like Jackie Chan" is:})$$

$$P(\text{negative}|\texttt{The sentiment of the sentence ‘‘I like Jackie Chan" is:})$$

Framing lots of tasks as conditional generation

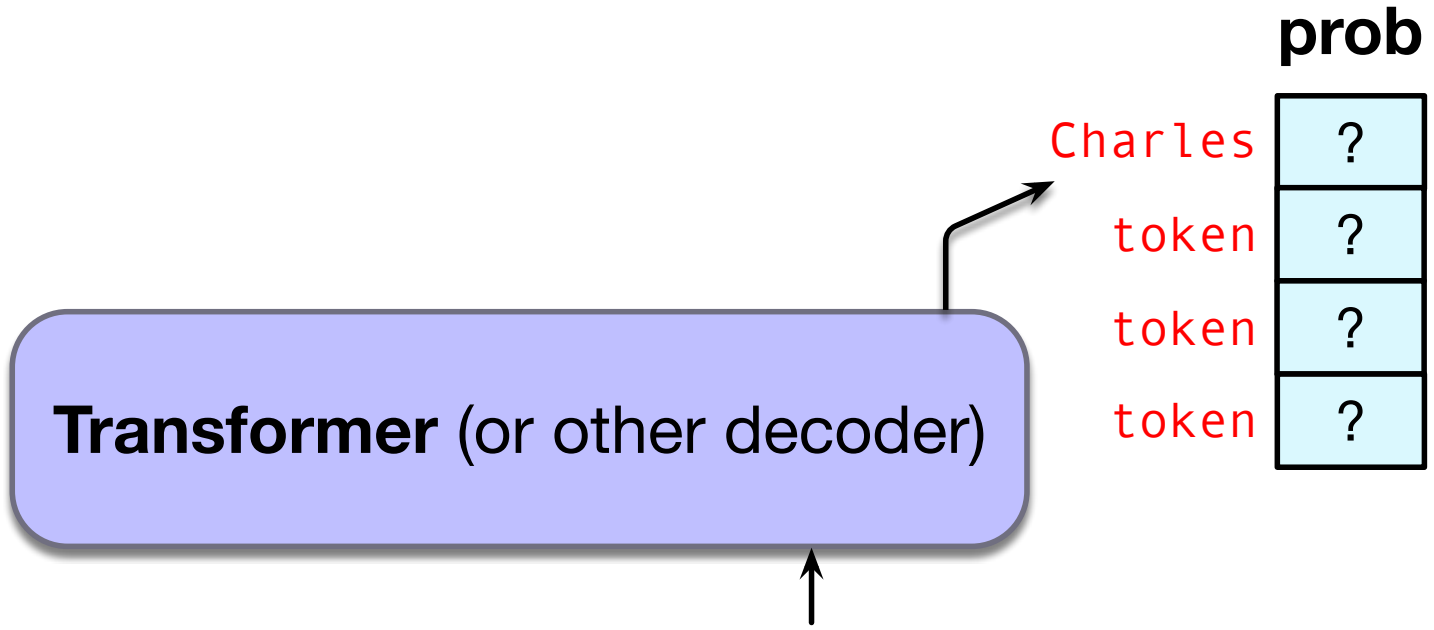
QA: “Who wrote The Origin of Species”

1. We give the language model this string:

```
Q: Who wrote the book ‘‘The Origin of Species"?  A:
```

2. And see what word it thinks comes next:

$$P(w|\texttt{Q: Who wrote the book ‘‘The Origin of Species"? A:})$$



Now we iterate:

$$P(w|\texttt{Q: Who wrote the book ``The Origin of Species"? A: Charles})$$

Large Language Models

Prompting

Conditional Generation with Prompts

Prompt: a text string that a user issues to a language model to get the model to do something useful by ***conditional generation***.

Prompt engineering: the process of finding ***effective prompts*** for a task.

Prompts

A **direct question**:

```
What is a transformer network?
```

Structured to fit pretraining data:

```
Q: What is a transformer network? A:
```

An **instruction**:

```
Translate the following sentence into Hindi: ‘Chop the garlic finely’.
```

Prompts can be very structured

Human: Do you think that “input” has negative or positive sentiment?
Choices:
(P) Positive
(N) Negative

Assistant: I believe the best answer is: (

Prompts can have demonstrations
also called IC/ICL examples

The following are multiple choice questions about high school computer science.

Let x = 1. What is x << 3 in Python 3?
(A) 1 (B) 3 (C) 8 (D) 16
Answer: C

Which is the largest asymptotically?
(A) O(1) (B) O(n) (C) O(n^2) (D) O(log(n))
Answer: C

What is the output of the statement “a” + “ab” in Python 3?
(A) Error (B) aab (C) ab (D) a ab
Answer:

2 demonstrations

Prompts are a learning signal

This is especially clear with ***demonstrations***.

But this is a different kind of learning than *pretraining*:

- **Pretraining** sets language model weights via gradient descent
- **Prompting** just changes the context and the activations in the network; no parameters change

Some people call this **in-context learning**—learning that improves model performance but does not update parameters.

Less confusing term would be **in-context examples**.

LLMs usually have a system prompt

<system>You are a helpful and knowledgeable assistant. Answer concisely and correctly.

This is automatically and silently concatenated to a user prompt.

- Intended to specify *persona*, *role*, *behavior constraints*.

<system> You are a helpful and knowledgeable assistant. Answer concisely and correctly.

<user> **What is the capital of France?**

System prompts can be long; 1700 words for Claude Opus4

Some extracts:

Claude should give concise responses to very simple questions, but provide thorough responses to complex and open-ended questions.

Claude is able to explain difficult concepts or ideas clearly. It can also illustrate its explanations with examples, thought experiments, or metaphors.

Claude does not provide information that could be used to make chemical or biological or nuclear weapons.

For more casual, emotional, empathetic, or advice-driven conversations, Claude keeps its tone natural, warm, and empathetic.

Claude cares about people's well-being and avoids encouraging or facilitating self-destructive behavior.

If Claude provides bullet points in its response, it should use markdown, and each bullet point should be at least 1-2 sentences long unless the human requests otherwise.

Large Language Models

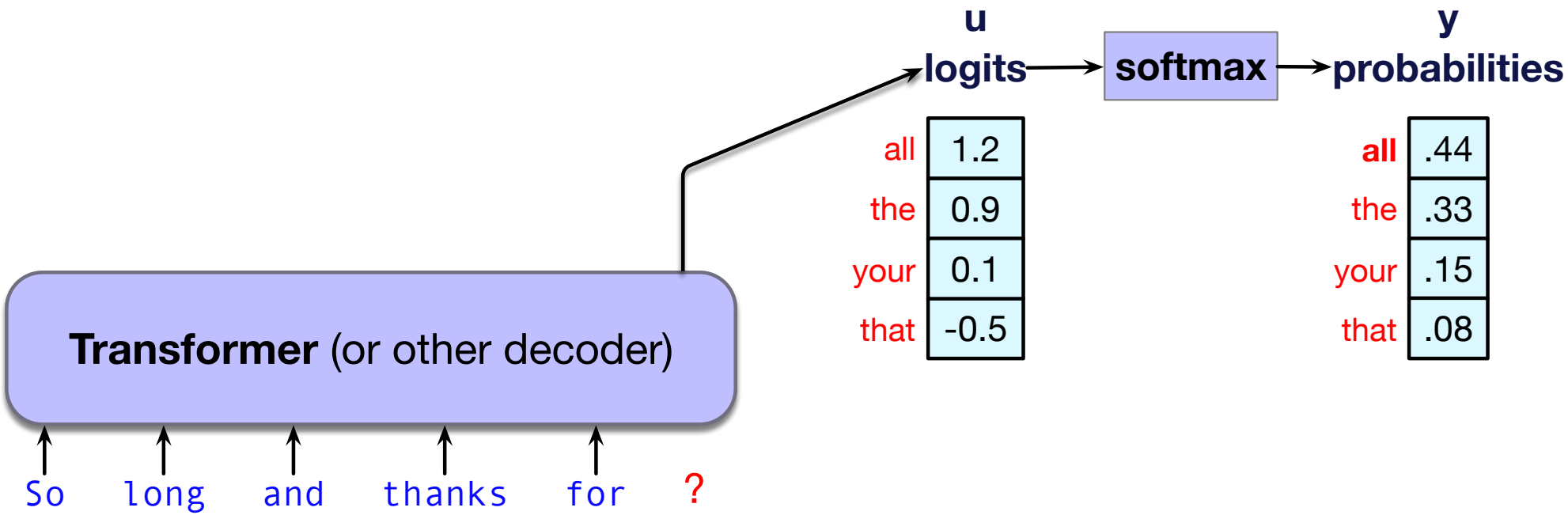
Sampling for LLM Generation

Where does token probability come from?

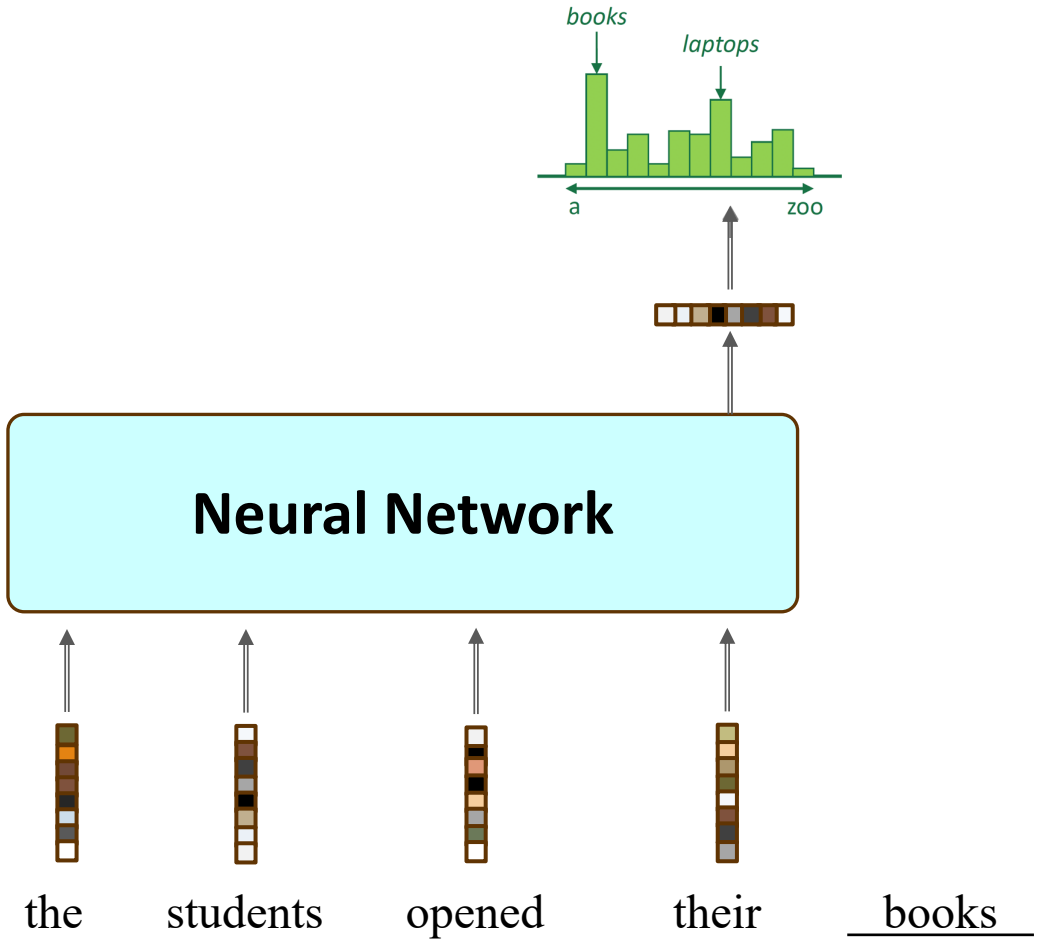
The internal networks for LLMs generate real-valued scores called **logits** for each token in the vocabulary.

Score vector **u** of shape $[1 \times |V|]$ is turned into a probability by *softmax*:

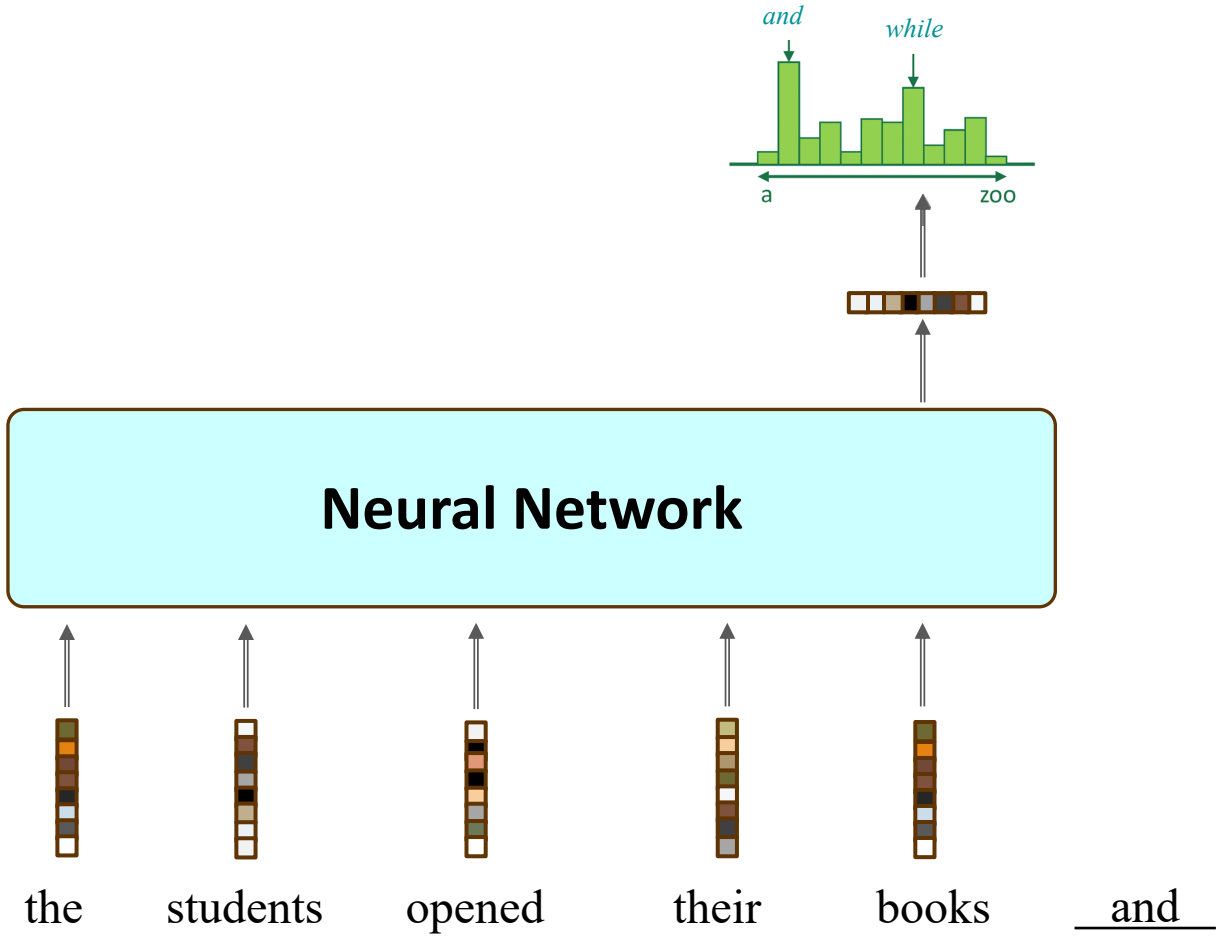
$$\mathbf{y} = softmax(\mathbf{u})$$



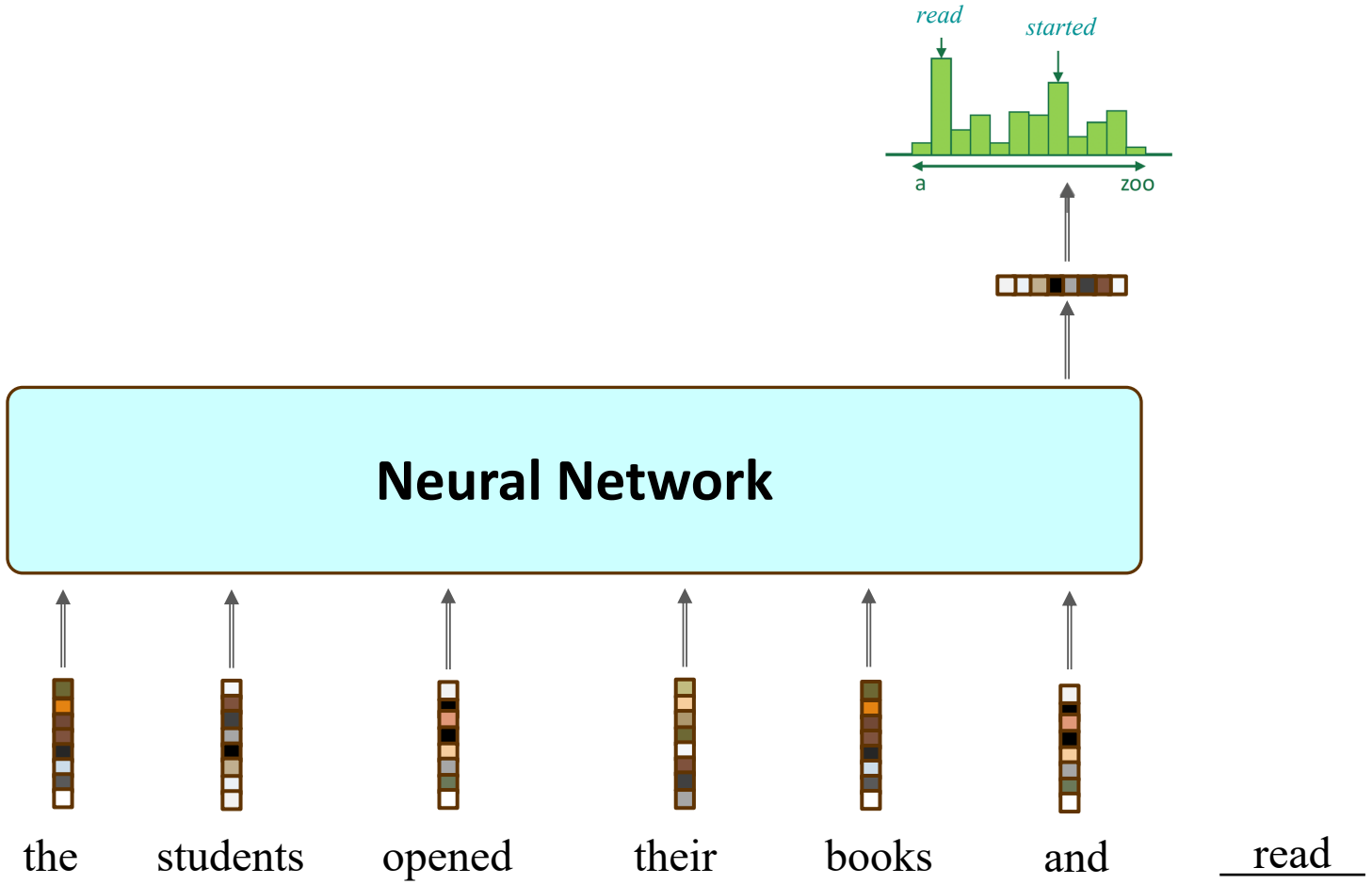
Neural Language Modeling: **Decoder**



Neural Language Modeling: **Decoder**



Neural Language Modeling: **Decoder**



Decoding

Decoding: the task of choosing a word to generate based on the model's probabilities, i.e., *sampling words*.

Autoregressive generation: Decoding from a model left-to-right, and repeatedly *choosing the next token conditioned on previously generated tokens*.

Greedy decoding

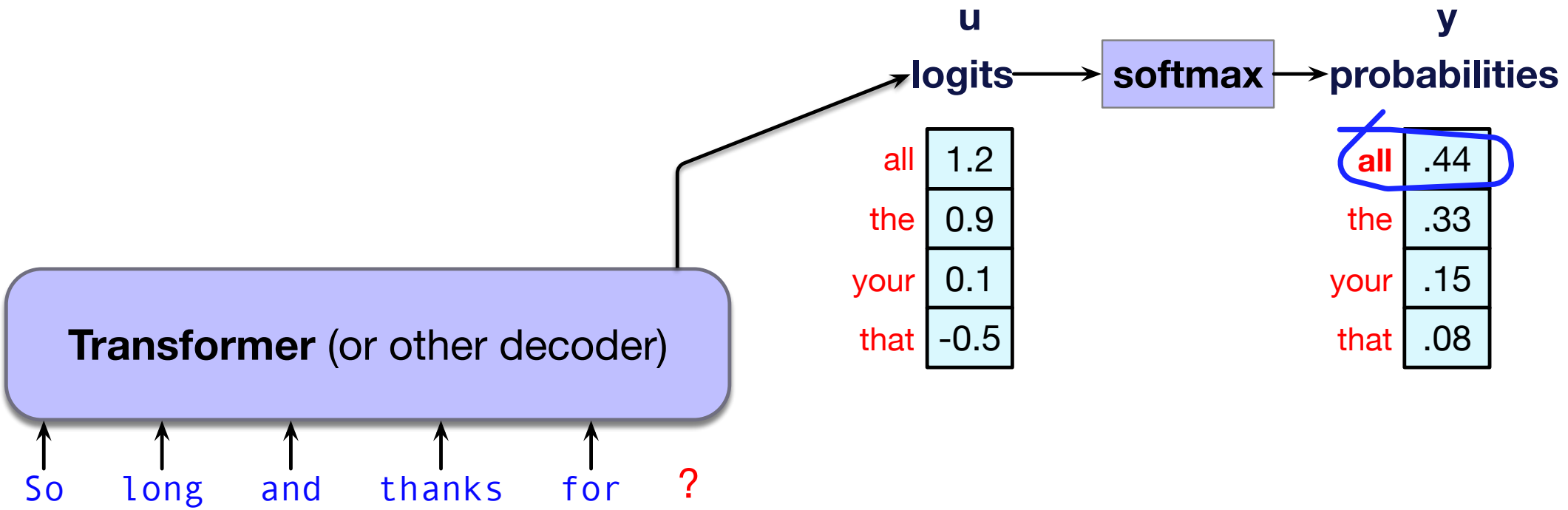
A **greedy algorithm** is one that makes a choice that is locally optimal:

- whether or not it will turn out to have been the best choice in hindsight.

Greedy decoding = simply generate the most probable word:

$$\hat{w}_t = \text{argmax}_{w \in V} P(w|\mathbf{w}_{<t})$$

Greedy decoding: choosing token that has maximum probability



Random sampling decoding

Sampling from a distribution means to choose random points according to their likelihood:

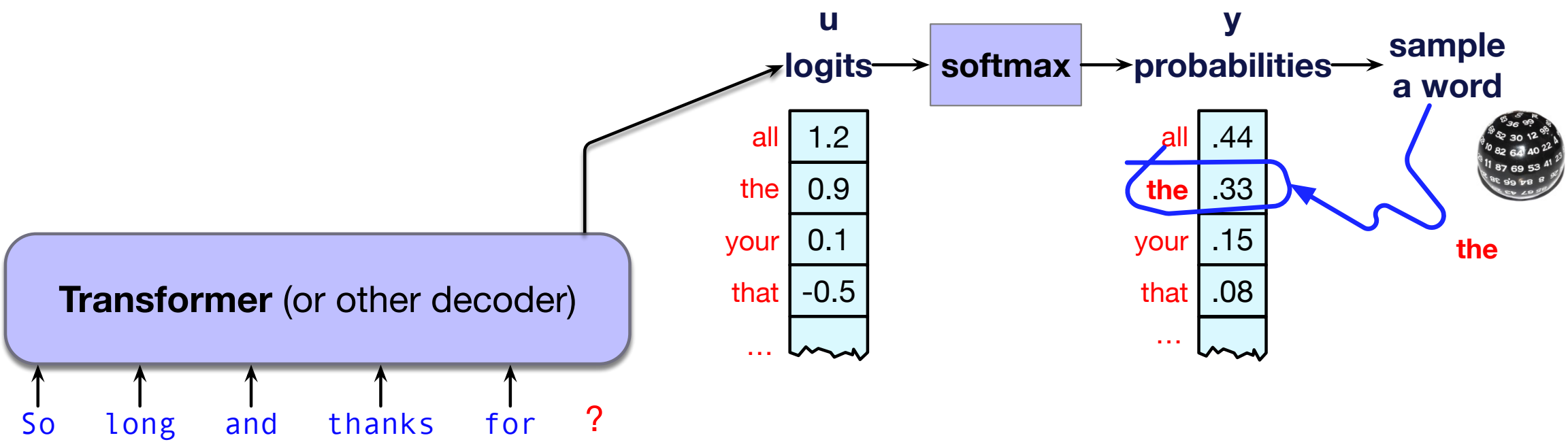
- **Sampling from an LM** means to choose the next token to generate according to the LM probability.

Multinomial sampling: We randomly select a token to generate according to its LM probability, conditioned on previous choices

Decoding:

- Sample token, append it to the input sequence.
- Repeat until EOS token is generated.

Random Sampling



Random sampling

$$
\begin{aligned}
&\mathrm{i} \leftarrow 1 \\
&w_i \sim \mathrm{p}(w) \\
&\textbf{while } w_i \text{ != EOS} \\
&\quad \mathrm{i} \leftarrow \mathrm{i} + 1 \\
&\quad w_i \sim \mathrm{p}(w_i \mid w_{<i})
\end{aligned}
$$

Greedy Decoding vs. Multinomial Sampling

Greedy Decoding useful when trying to get the “best” answer from the LLM, in one autoregressive generation.

- Coding, math, …
- Because the tokens it chooses are extremely predictable, the result can be **generic** and **repetitive**.

Multinomial Sampling useful when:

- Prefer text that is more diverse.
- Need to generate multiple sequences, then select the best one:
  - Coding: generate multiple programs, select the one that passes all tests.

Alas, random sampling doesn't work very well

Even though random sampling mostly generate sensible, high-probable words,

There are **many odd, low-probability words** in the tail of the distribution:

- Each one is low- probability, but added up they constitute a **large portion of the distribution**.
- So they get picked enough to generate weird sentences.

Factors in word sampling: **quality** and **diversity**

Emphasize **high-probability** words

+ **quality**: more accurate, coherent, and factual.

- **diversity**: boring, repetitive.

Emphasize **middle-probability** words

+ **diversity**: more creative, diverse.

- **quality**: less factual, incoherent

Temperature sampling

Reshape the probability distribution to either:

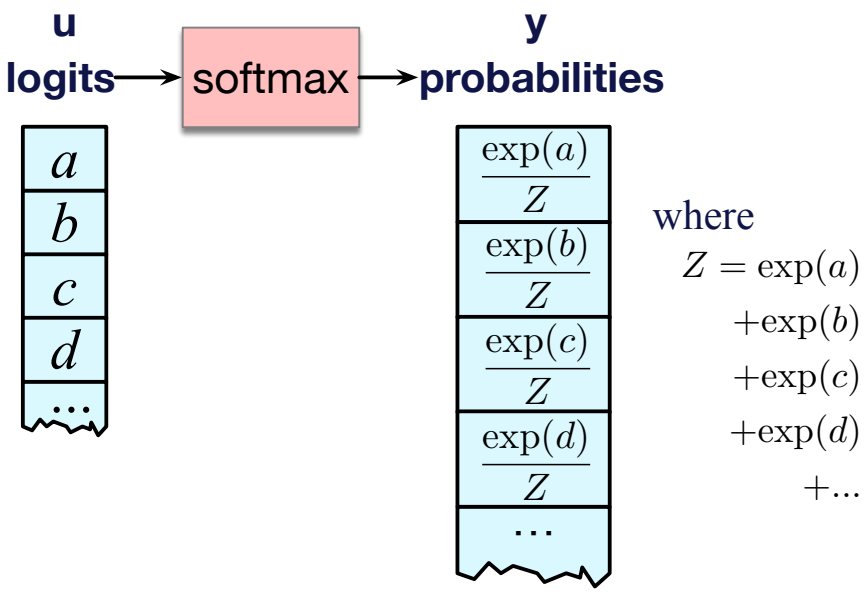
- increase/decrease the probability of the high probability tokens.
- decrease/increase the probability of the low probability tokens.

Divide the logit by a temperature parameter τ before passing it through the softmax.

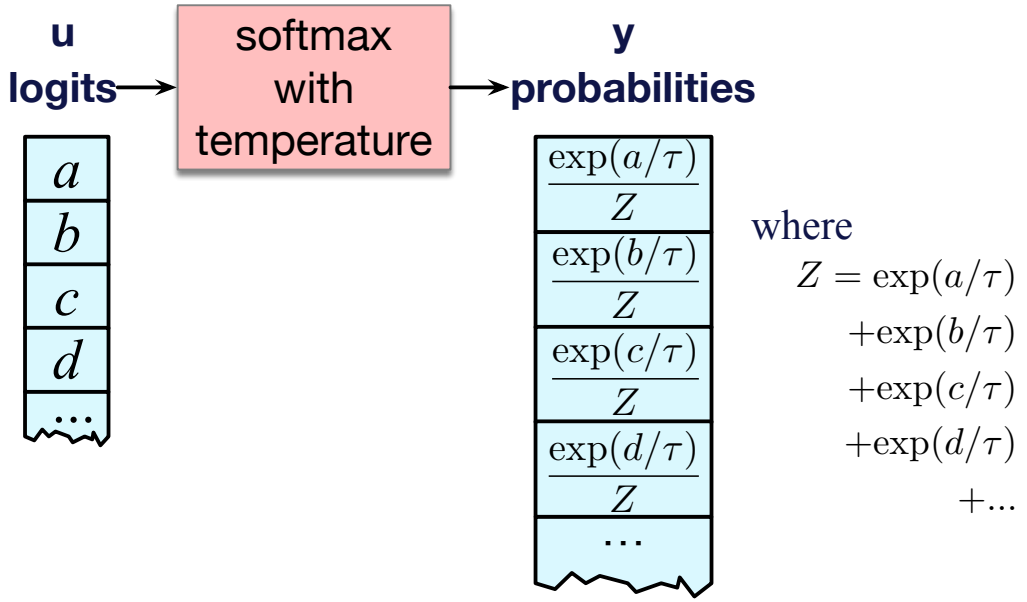
- Instead of: ~~$\mathbf{y} = \mathrm{softmax}(u)$~~
- We compute: $\mathbf{y} = \mathrm{softmax}(u/\tau)$

$0 \leq \tau \leq 1$ vs. $1 < \tau \leq \infty$

Temperature sampling



(a)



(b)

Temperature sampling

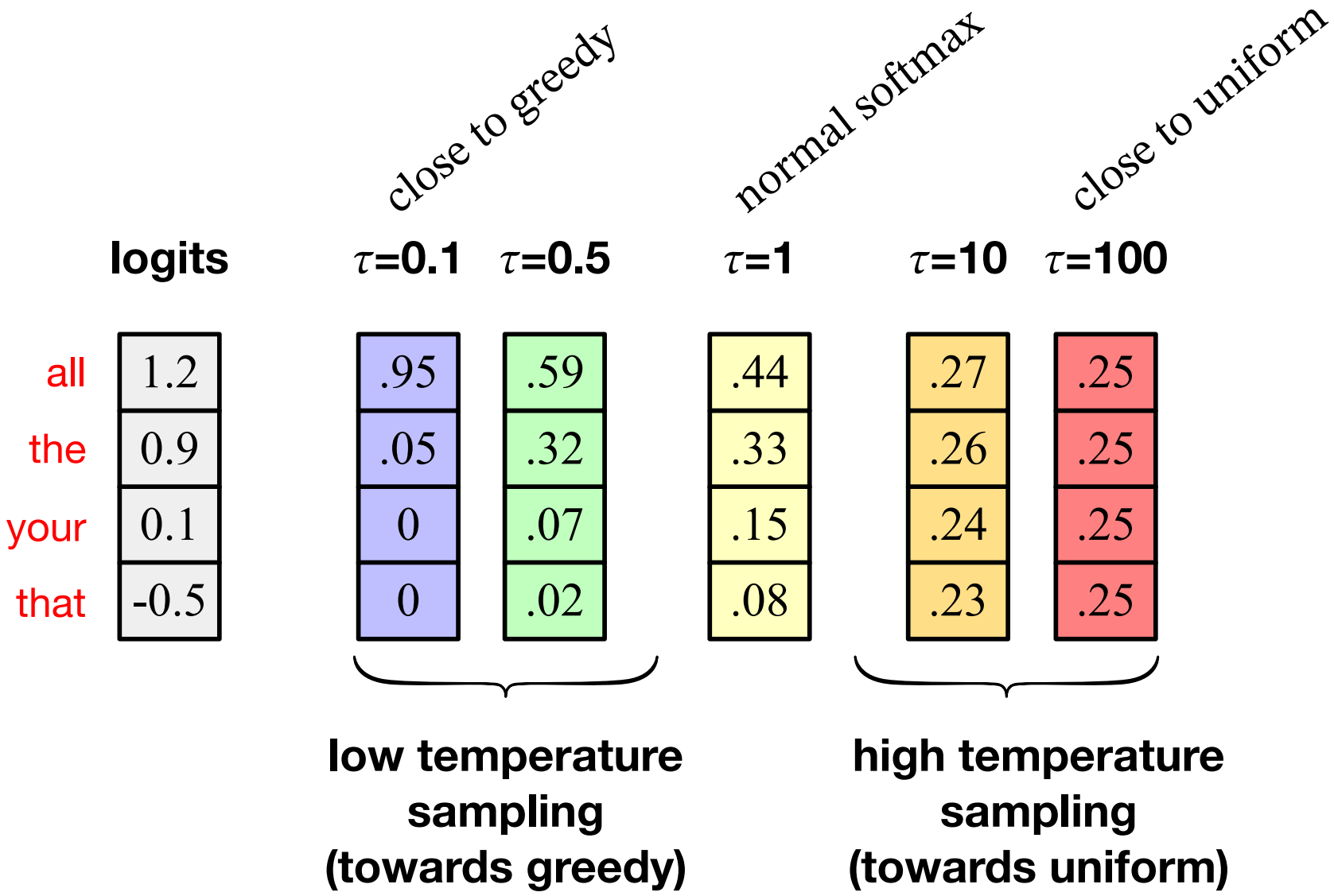
$$\mathbf{y} = \text{softmax}(u/\tau)$$

$0 \leq \tau \leq 1$ vs. $1 < \tau \leq \infty$

Why does this work?

- When τ is close to 1 the distribution doesn't change much.
- When τ is close to 0, larger scores are amplified more by softmax:
  - Softmax pushes high values toward 1 and low values toward 0.
  - Large inputs pushes high-probability words higher and low probability word lower, making the distribution more greedy.
  - As τ approaches 0, the probability of most likely word approaches 1.
- When τ is close to ∞, all scores becomes closer to 0 and equal to each other.

softmax output with temperature τ



Temperature sampling comes from **Thermodynamics**

At high temperature ($\tau > 1$), a system is flexible and can explore many possible states:

- A system at lower temperature is likely to explore a subset of lower energy (better) states.

In **low-temperature sampling**, ($\tau \leq 1$) we smoothly:

- increase the probability of the most probable words
- decrease the probability of the rare words.

Top-K vs. Top-p (Nucleus) Sampling

Top-K sampling:

- Sample only from the top-*K* probability tokens.

Top-p sampling:

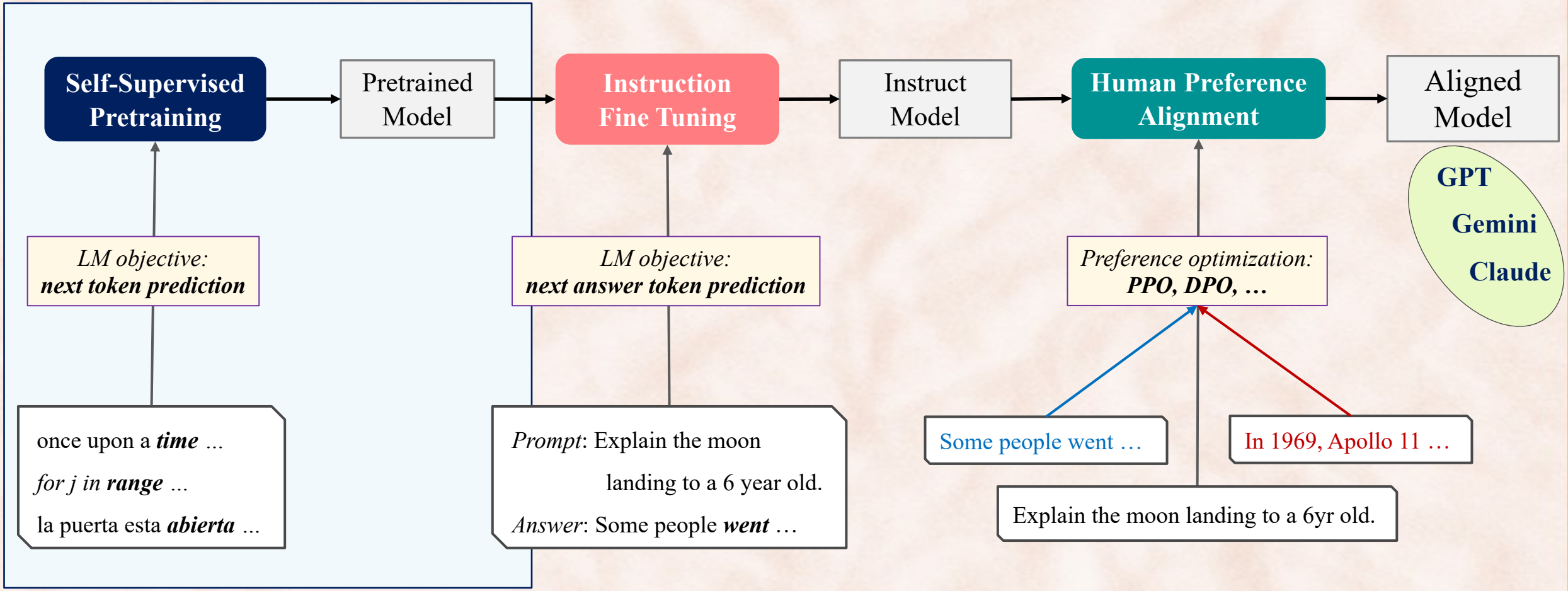
- Sample only from the top probability tokens whose total probability mass is equal to *p*.

In bot top-K and top-p sampling: renormalize the token probabilities to sum up to 1.

Large Language Models

Training Large Language Models

Training LLMs: Pretraining, Instruction tuning, Alignment



Pretraining: Language Modeling (LM) Objective

- **Causal** Language Modeling:
  - **Predict the next word in a sequence**:
    - AI systems use machine _____
      - eat?
      - learning?
      - frogs?
      - …
    - **The LM estimates P(word | $word_{-1}$, $word_{-2}$, ...)**
      - we want P(learning | machine, use) >> P(about | machine, eat).
  - **Decoder** neural architectures are widely used to train LMs:
    - GPT, Gemini, Llama, Grok, Mixtral, Claude, …
- Pretraining = training to **predict next word** given on a very large corpus of text.

Pretraining: Self-supervised training algorithm

We train to predict the next word => classification:

1. Take a corpus of text
2. At each time step *t*
   i. ask the model to predict the next word (distribution).
   ii. update the model using gradient descent to minimize the error in this prediction.

Self-supervised because it just uses the next word as the label.

Intuition of Language Modeling objective

Same loss function as in classification, **cross-entropy loss**:

- We want the model to assign a high probability to true word ***w***
- = want loss to be high if the model assigns too low a probability to **w**

CE Loss: The *negative log probability* that the model assigns to the true next word **w**

- If the model assigns too low a probability to w
- We move the model weights in the direction that assigns a higher probability to w

Cross-entropy loss for language modeling

$$H(p, q) = -\mathrm{E}_p[\log q] = -\sum_{x \in \mathcal{X}} p(x) \log q(x).$$

CE loss: "difference" between correct probability distribution and predicted distribution:

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

The correct distribution $\mathbf{y}_t$ is 1 for the actual next word, and 0 for the others. In the sum, all terms get multiplied by 0, except one: the *logp* the model assigns to the correct next word *w*:

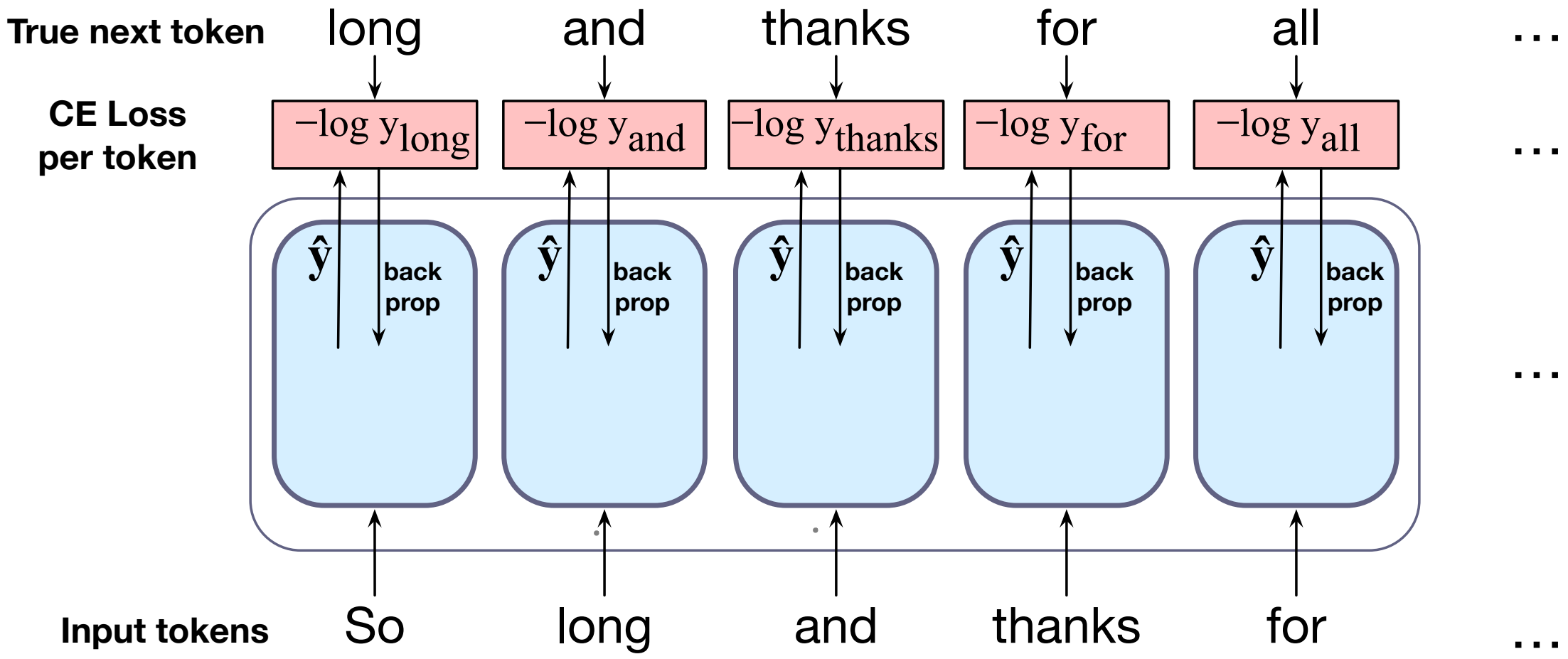
$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

Teacher Forcing

- At each token position t, model sees all the **correct** tokens so far $\boldsymbol{w_{1:t}}$:
  - Computes loss (–log probability) for the next token w_{t+1}

- At next token position t+1 we ignore what model predicted:
  - Instead, we take the **correct** word w_{t+1}, add it to context, move on

Pretraining an RNN/Transformer language model



LLMs are mainly pretrained on the web

https://commoncrawl.org

Common Crawl repository:

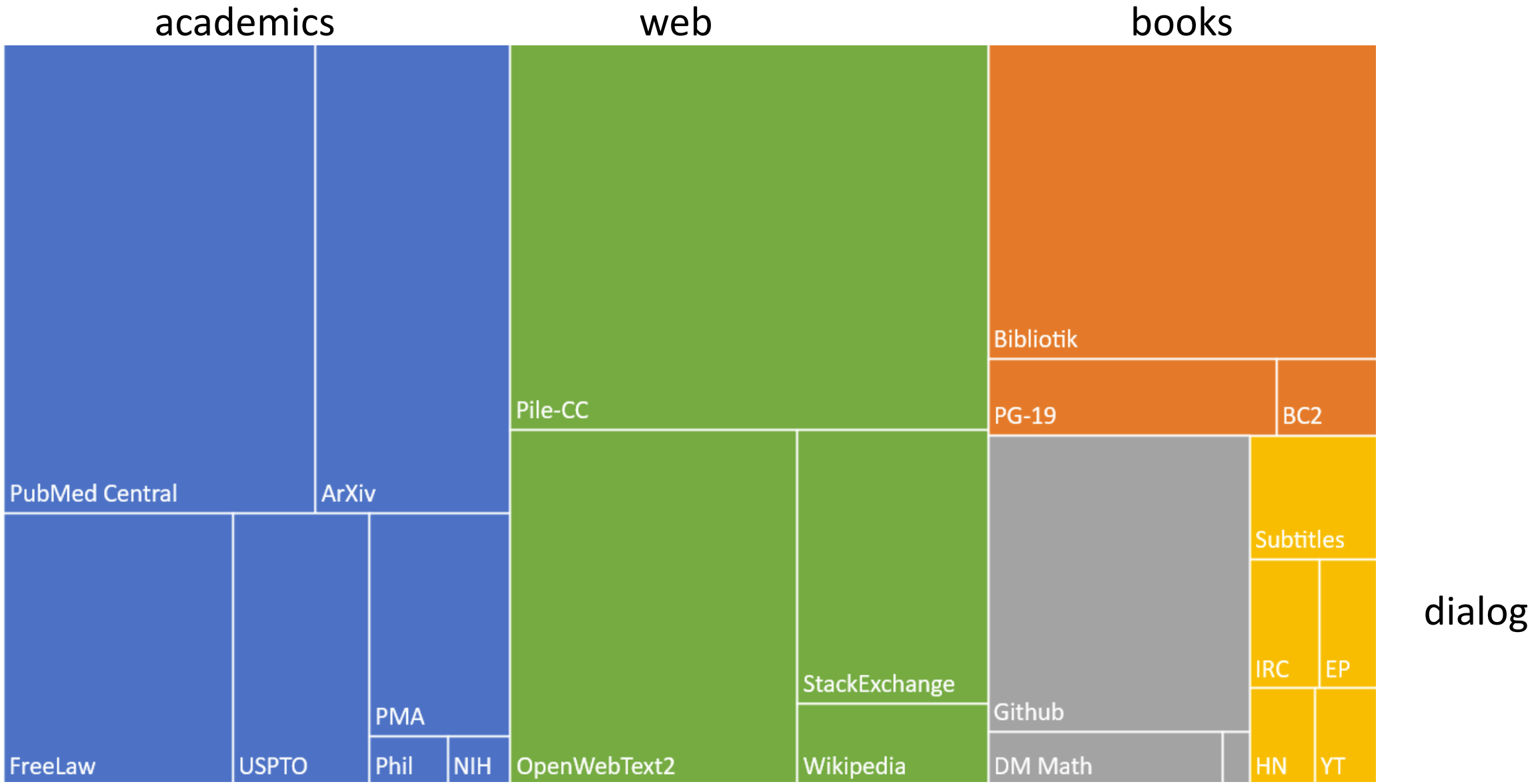
- snapshots of the entire web produced by the non- profit Common Crawl with billions of pages.

Colossal Clean Crawled Corpus (C4; Raffel et al. 2020):

- 156 billion tokens of English, filtered.
  - Mostly patent text documents, Wikipedia, and news sites.

The Pile: a pretraining corpus

https://pile.eleuther.ai



Filtering for quality and safety

Quality is subjective:

- Many LLMs attempt to match Wikipedia, books, particular websites.
- Need to remove boilerplate, adult content,
- Deduplication at many levels (URLs, documents, even lines).

Safety also subjective

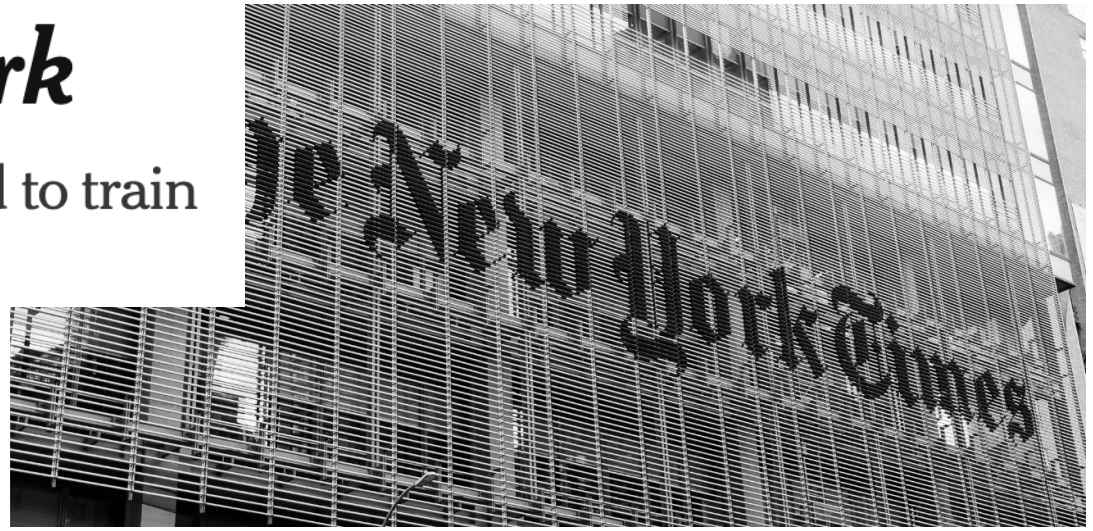
- Toxicity detection is important, although that has mixed results.
- Can mistakenly flag data written in dialects like African American English.

There are problems with scraping from the web

Authors Sue OpenAI Claiming Mass Copyright Infringement of Hundreds of Thousands of Novels

The Times Sues OpenAI and Microsoft Over A.I. Use of Copyrighted Work

Millions of articles from The New York Times were used to train chatbots that now compete with it, the lawsuit said.



Potential problems with scraping from the web

Copyright: much of the text in these datasets is copyrighted

- Not clear if fair use doctrine in US allows for this use
- This remains an open legal question across the world

Data consent:

- Website owners can indicate they don't want their site crawled

Privacy:

- Websites can contain private IP addresses and phone numbers.

Skew:

- Training data is disproportionately generated by authors from the US which probably skews resulting topics and opinions

Plagiarism in generated output

AudioLM: A Language Modeling Approach to Audio Generation

- https://google-research.github.io/seanet/audiolm/examples

Piano continuation

AudioLM is not limited to modeling speech. It can also learn to generate coherent piano music continuations, despite being trained on piano music without any symbolic representation. We also show the continuations produced by a version of the model trained exclusively on the acoustic tokens. These continuations are much less coherent, stressing the importance of the semantic tokens in our framework. The 4-second prompts come from the test split of MAESTRO dataset.

Original	Prompt	Continuation by acoustic-only model	Continuation by AudioLM
0:10 / 0:20	0:04 / 0:04	0:00 / 0:20	0:20 / 0:20

AudioLM plagiarized the 3rd movement of the Moonlight sonata, the motif starting at minute 10:46.

https://youtu.be/4591dCHe_sE?t=644

Plagiarism in generated output

NotebookLM has a podcast feature:

https://notebooklm.google/audio

Former "Morning Edition" host David Greene alleges in a lawsuit that Google patterned the "voice" of one of its AI products after his without permission.

https://www.npr.org/2026/02/17/nx-s1-5716055/former-morning-edition-host-accuses-google-of-stealing-his-voice-for-ai-product

Large Language Models

Finetuning

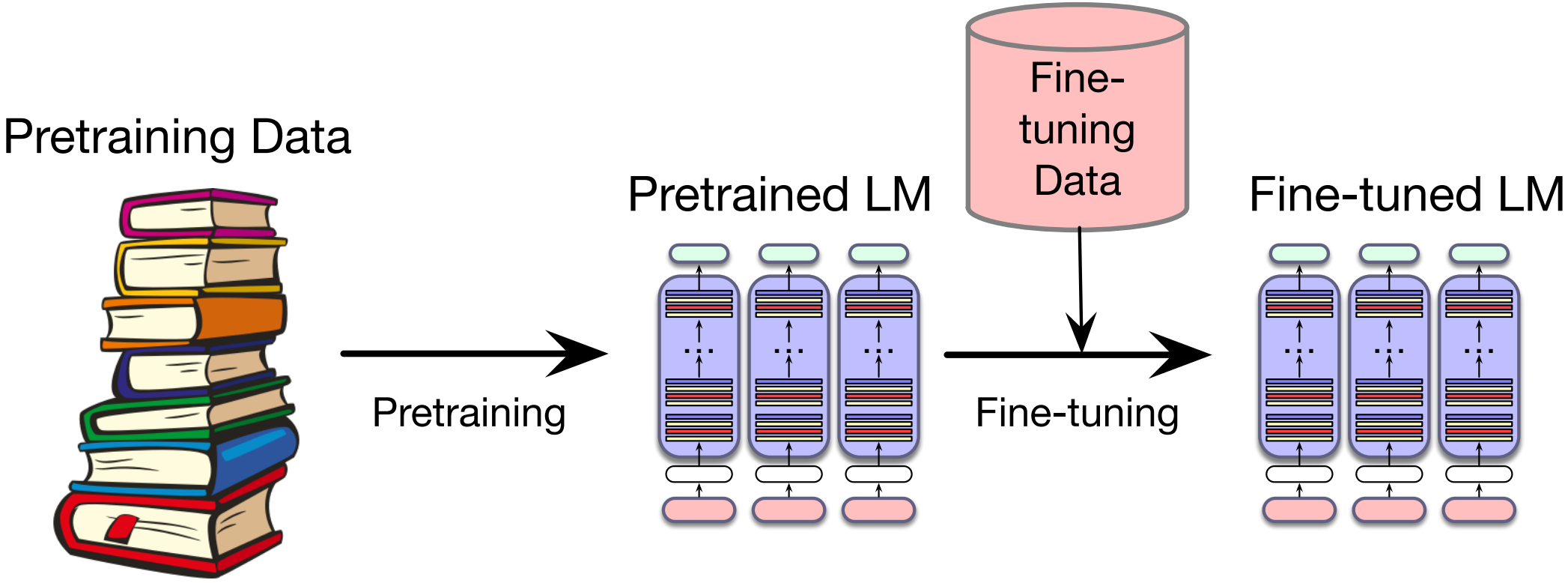
Finetuning for adaptation to new domains

What happens if we need our LLM to work well on a domain it didn't see in pretraining?

Perhaps some specific **medical** or **legal** domain?

Or maybe a multilingual LM needs to see more data on some language that was rare in pretraining?

Finetuning


Pretraining Data
Pretraining
Pretrained LM
Fine-
tuning
Data
Fine-tuning
Fine-tuned LM

"Finetuning" means 4 different things

We'll discuss 1 here, and 3 in later lectures.

In all four cases, **finetuning** means:

Taking a pretrained model and further adapting

some **or** ***all*** **of its parameters to some new data.**

1. Finetuning as "continued pretraining" on new data

Further train **all** the parameters of model on new data:

- Using the same method (word prediction) and loss function (cross-entropy loss) as for pretraining.
- As if the new data were at the tail end of the pretraining data.

- Hence sometimes called **continued pretraining**.

Knowledge Distillation

Distillation is a special type of fine-tuning:

- A "*student*" model is trained to reproduce the behavior and performance of a complex "*teacher*" model.
- Multiple approaches, depending on what is available from teacher:
  1. Train student to match the softmax distribution of the teacher:
     - Minimize **KL-divergence**($P_{student}$, $P_{teacher}$)
  2. Train student on <prompt, teacher output> pairs from teacher.

https://www.anthropic.com/news/detecting-and-preventing-distillation-attacks

Supplemental Reading

- Chapter 7 on LLMs from J & M, up to and including section 7.4.